Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

Oleksii Abramenko

Graph signal sampling via reinforcement learning

Master's Thesis
Espoo, October 20, 2018

Supervisor: Professor Alexander Jung

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

ABSTRACT OF MASTER'S THESIS

Author:	Oleksii Abramenko		
Title:	Graph signal sampling via reinforcement learning		
Date:	October 20, 2018	**Pages:**	59
Major:	Machine Learning and Data Mining	**Code:**	SCI3044
Supervisor:	Professor Alexander Jung		

Graph signal sampling is one the major problems in graph signal processing and arises in a variety of practical applications, such as data compression, image denoising and social network analysis. In this thesis we formulate graph signal sampling as a reinforcement learning (RL) problem, which unleashes powerful methods developed recently within RL. Within our approach the signal sampling is carried out by an agent which crawls over the empirical graph and selects the most relevant graph nodes to sample, i.e., determines the corresponding graph signal values. Overall, the goal of the agent is to select signal samples which allow for the smallest graph signal recovery error. The behavior of the sampling agent is described using a policy which determines whether or not a particular node should be sampled. The policy is continuously adjusted which implies an inherent robustness to changes in the data structure.

We propose two RL-based sampling algorithms and evaluate their performance by means of illustrative numerical experiments. After this we conduct elaborative discussion on strengths and weaknesses of the proposed solutions and explain phenomenons observed during simulations.

Based on the simulation results, we identify some major challenges related to application of RL to graph signal sampling and propose possible solutions leading to prospects for the future work.

Keywords:	machine learning, reinforcement learning, graph signal sampling, convex optimization, multi-armed bandit, policy gradient, complex networks
Language:	English

Acknowledgements

I would like to express my gratitude to Professor Alexander Jung who has enthusiastically supervised me for the last two years and not only introduced me to various machine learning methods but also greatly expanded my understanding of this field as a whole. Membership in his research group "Machine learning for Big Data" has allowed me to obtain priceless research experience while working on relevant machine learning problems.

I am highly indebted to all professors and staff of Aalto University, who have demonstrated exceptional level of competence and maintained high academic standards ensuring that their students get access to the state-of-art knowledge and forefront educational services.

Finally, I would like to thank to my family and friends who have been very supportive of me all the way through my studies and provided their help when I needed it.

Espoo, Finland
October 20, 2018 Oleksii Abramenko

Abbreviations and Symbols

GSP	Graph Signal Processing
SLP	Sparse Label Propagation
LASSO	Least Absolute Shrinkage and Selection Operator
TV	Total Variation
MDP	Markov Decision Processes
MAB	Multi-Armed Bandit
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
DQN	Deep Q-Network
MSE	Mean Squared Error
SBM	Stochastic Block Model
RWS	Random Walk Sampling
URS	Uniform Random Sampling

$\mathbf{x}$	a vector
x	a scalar
$\mathbf{X}$	a matrix
$x[i]$	i-th element of the vector $\mathbf{x}$
$\|\mathbf{x}\|_{\mathrm{TV}}$	total variation
$\|\mathbf{w}\|_1$	l_1 norm
$\|\mathcal{M}\|$	cardinality
$\mathbf{x}^2$	element-wise square of the vector
$\mathbf{x}/\mathbf{y}$	element-wise division of two vectors
$\arg\max_x f(x)$	find such x that maximizes $f(x)$

Contents

Chapter 1

Introduction

Daily activities of people all across the globe generate massive amounts of data of different forms, from text messages and audio/video recordings to the specialized data produced by the industrial applications and even scientific experiments. Due to the new technologies and services, including broadband mobile networks and cloud storages, data generation rate expands exponentially and in unprecedented scale. A special place in this Big Data World is occupied by the structured data, or data which can be represented as a network of elements, where close or similar in some sense elements are connected. One striking example of the data with such an underlying structure is a social network (i.e. Facebook), where each user has a connection to several other "friend" users. Other examples of structured data may be found in bioinformatics and chemistry (networks of atoms forming molecules and networks of molecules forming proteins), astronomy (representing the Universe as a network of planets, stars, galaxies) and many other areas.

Intrinsic network structure of numerous real-world problems allows us to analyze them using graph theory by representing network elements as nodes of a graph and connections between them as edges. Such a representation provides a set of powerful mathematical instruments which can be applied for solving relevant problems. For example, friend recommendation in social networks, such as Facebook, can be seen as a link prediction problem [30], which is a classical problem arising in the analysis of graphs. It is important to note that such a structured network data is often redundant in many ways and can be represented in a compressed or sparse way. This can be particularly useful for dealing with data compression and recovery problems, where the whole empirical graph can be recovered from relatively small number of samples. In this context, first major challenge is to select the most representative samples, i.e., nodes which carry maximum information about the whole network-structured dataset. Another issue is how to recover the

required information from this relatively small number of observations. The latter problem has been extensively studied over the recent years and several convenient and efficient mathematical formulations allowing for straightforward solution have been derived. Nowadays, majority of advances are mostly focusing on improving performance and scalability of the previously proposed recovery methods. On the other hand, solution to the former problem is not straightforward (details in Section 2.2) and requires new approaches.

One of these fundamentally new approaches is reinforcement learning, a machine learning paradigm which has acquired much attention and shown great success in approaching problems which have not been solved efficiently for decades. For instance, in 2010 researchers proposed Deep Q-Network (DQN) algorithm [33] for playing popular Atari games. The success of the novel RL architecture has inspired applications of RL in a variety of new fields including natural language processing [29], robotics [54], computer vision [52] and continuous control [31].

The main objective of this thesis is to interpret graph sampling as a reinforcement learning problem. In particular, we interpret online sampling algorithm as an artificial intelligence agent which chooses the nodes to be sampled on-the-fly. The behavior of the sampling agent is represented by a policy over a discrete set of different actions which are at the disposal of the sampling agent in order to choose the next node at which the graph signal is sampled. The ultimate goal is to learn a sampling policy which determines signal samples that allow for a small reconstruction error. Our goal is to study classical RL and recently proposed deep reinforcement learning (DRL) methods and investigate their applicability to the graph signal sampling problem. We strongly believe that usage of reinforcement learning and especially deep reinforcement learning approaches may not only bring the graph signal sampling to the qualitatively new level but can also open new horizons for the graph signal processing (GSP) in general.

Thesis contribution

This thesis models graph signal sampling as a RL problem. Based on this modelling we devise novel graph signal sampling algorithms. We discuss design of the proposed algorithms, state-action space and reward function choice, and formulate difficulties associated with usage of RL for graph signal sampling. The usefulness of the proposed sampling methods is verified by means of illustrative numerical experiments.

Structure of the Thesis

In Chapter 2 we provide a brief review of some main concepts and results within graph signal processing, with emphasis on machine learning problems. Chapter 3 studies theoretical foundations of RL and describes key algorithms which will be used in the subsequent chapters. Our main contribution is in Chapter 4 where we design two RL algorithms for graph signal sampling. In the following Chapter 5 we conduct numerical experiments, which are discussed and analyzed in Chapter 6. Chapter 7 concludes all the work done within the master's thesis by listing key elements of the research and highlighting its relevance.

Chapter 2

Machine learning for graph signal processing

In this chapter we present fundamentals of the GSP and the role machine learning plays in recovering signals defined over graphs. We start with formulating general mathematical model of the GSP, including all necessary theoretical assumptions, followed by the description of methods and approaches used to obtain sampled and recovered signals.

2.1 Graph signal processing fundamentals

We consider datasets which can be represented using an empirical graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$. The graph $\mathcal{G}$ is an undirected simple graph (with no self-loops or multi-edges). The nodes $i \in \mathcal{V}$ of the empirical graph correspond to the data points, whereas the edges $\{i, j\} \in \mathcal{E}$ connect data points which are similar. The extent of similarity between data points is quantified by the weight matrix $\mathbf{W} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$. We now associate each data point $i \in \mathcal{V}$ with the feature vector $\mathbf{f}_i \in \mathbb{R}^d$ and label $x[i] \in \mathbb{R}$. These data labels induce a signal $\mathbf{x} : \mathcal{V} \to \mathbb{R}$ defined over graph $\mathcal{G}$ (see Figure 2.1). The similarity matrix $\mathbf{W}$ can be obtained in numerous ways, i.e., can be provided by a human expert or calculated automatically based on the distances between data points in the feature space, for example by using inverse multiquadric kernel [38]:

$$w_{ij} = \frac{1}{\sqrt{\|\mathbf{f}_i - \mathbf{f}_j\|^2 + c^2}},$$

where c is a regularization constant.

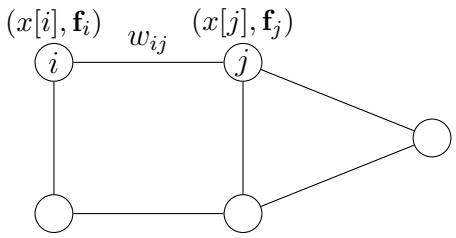


Figure 2.1: Representing a dataset via an empirical graph. Notation: $x[i]$, $x[j]$ – graph signal values; $\mathbf{f}_i$, $\mathbf{f}_j$ – node feature vectors; w_{ij} – edge weight.

Classical examples of structured graphs are grid and chain graphs (see Figure 2.2) widely used in modern signal processing applications. For instance, a 2D image can be easily represented as a grid graph, with pixels being nodes and similarities between pixels being edges. The same reasoning applies to the chain graph representation which can model time series or one-dimensional signal of any nature.

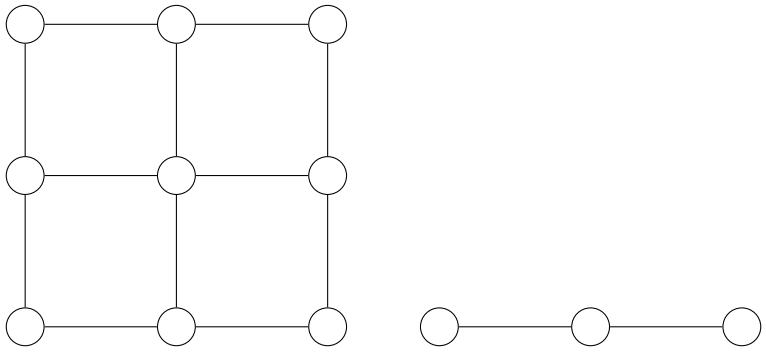


Figure 2.2: Examples of network-structured datasets: (a) – grid graph representing a 2D image, (b) – chain graph representing a time series.

In some applications the edges of the empirical graph are naturally defined, e.g., via friendship relations in social networks. However, sometimes it is useful to learn the network structure in a data-driven fashion. One example of such a task is a graphical model selection problem [19, 21, 36] which comes down to learning relationships between random variables in order to build conditional independence graph. Besides that, network structure learning is used in many areas. For instance, Tan et. al. [42] propose data-driven network structure learning for multi-label image classification and the study [50] considers using graph learning to solve the scheduling problem.

This thesis mainly revolves around graph signal sampling and recovery which are considered to be key problems in the GSP field. Extending sampling of time signals (e.g. audio wave signals), graph signal sampling comes down to selecting a small subset of graph nodes whose signal values are representative for the entire graph signal. The dual problem to graph signal sampling is graph signal recovery. It is somewhat equivalent to the signal interpolation in the classical digital signal processing (DSP) and aims at determining signal values of all nodes based on signal values of nodes in the sampling set. Graph signal sampling and recovery are used together in many applications including data compression (compressing a signal using sampling and decompressing using recovery) and semi-supervised learning [2, 20]. Figure 2.3 depicts example of sampling and recovering an image defined over grid graph (see Figure 2.2a). In this example we obtain sampled representation by taking 25 % of all pixels uniformly at random and use it as an input to the recovery procedure described in the upcoming Section 2.3. It is worth mentioning that the recovered image does not match to the original one perfectly and the quality of recovery powerfully depends on the sampling set choice. Furthermore, if we take 100 random sampling sets, perform recovery based on them, for each case measure mean squared error of recovery (see Section 2.2) and plot its histogram (see Figure 2.4), we can note that the difference in MSE between the best (≈ 205) and the worst (≈ 270) sampling sets choices amounts to approximately 30 %.

(a) (b) (c)

Figure 2.3: Example of sampling and recovering graph signal defined over grid graph: (a) – original graph signal, (b) – sampled graph signal, (c) – recovered graph signal. Image “Lenna” source: http://computervision.wikia.com/wiki/File:Lenna.jpg

Provided example illustrates that drawing samples to the sampling set randomly does not guarantee that the obtained sampling set will allow to reconstruct the signal with maximal quality (minimal MSE of recovery),

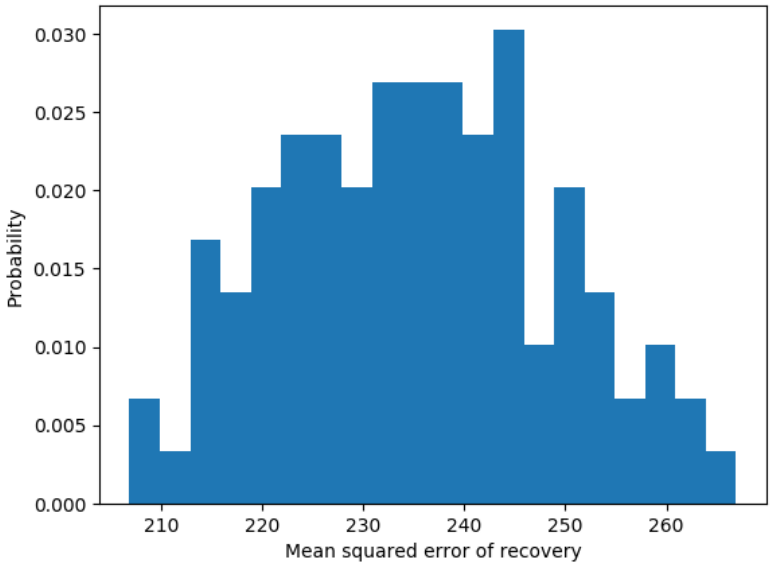


Figure 2.4: Histogram of mean squared error of graph signal recovery based on 100 random sampling sets.

therefore usage of advanced methods for finding such (sub)optimal sampling sets is necessary. In the forthcoming sections we will review existing methods for graph signal sampling and recovery and consider the most important of them in details.

2.2 Sampling graph signals

Graph signal sampling is the problem of finding a sampling set $\mathcal{M} := \{i_1, \ldots, i_M\} \subseteq \mathcal{V}$ of the predefined size $M = |\mathcal{M}|$. Since acquiring signal values is often expensive (requiring manual labor), the sampling set is typically much smaller than the overall dataset, i.e., $M = |\mathcal{M}| \ll N$. In GSP applications we often aim at selecting such a sampling set which would lead to a minimum reconstruction error. For numeric signal values (arising in regression problems), a popular measure of the reconstruction error is the mean squared error (MSE) between recovered (conditioned on the sampling set $\mathcal{M}$) $\hat{\mathbf{x}}^{\mathcal{M}}$ and true graph signals $\mathbf{x}$:

$$MSE := \frac{1}{N} \sum_{j \in \mathcal{V}} (x[j] - \hat{x}^{\mathcal{M}}[j])^2$$

Note that the MSE depends on both, the sampling set $\mathcal{M}$ and the recon-

struction method which produces $\hat{\mathbf{x}}^{\mathcal{M}}$.

Graph signal sampling is related to the famous sensor selection problem [18] which considers selection of a sensor set providing the most accurate estimate of some quantity. This problem [6] as well as graph signal sampling [49] have been proven to be NP-hard, which means that in order to find optimal (in terms of minimal reconstruction error) sampling set of size M, it is required to test all $\binom{N}{M}$ combinations. Obviously that in the big data environment such an approach quickly becomes computationally intractable. Even for relatively small datasets with $N = 100000$ elements and $M = 1000$ samples, number of possible sampling set combinations reaches astronomical order of 10^{500}. Within broad sensor selection context there is a variety of relevant methods including branch and bound search [27], convex relaxation [18], heuristic-based methods [51] and others. It should be mentioned that all these methods are either approximate or unable to deliver optimal solution in polynomial time.

Within the graph signal processing framework there is a massive amount of research which tries to address graph signal sampling problem from different perspectives. For instance, the study [9] dives deep into fundamentals of sampling theory of the bandlimited graph signals where authors investigate performance of several graph sampling algorithms, such as uniform random sampling (URS), experimentally designed sampling and active sampling, and establish error bounds for them. Another study [43] proposes completely different approach by handling sampling in the frequency domain and manipulating with Discrete Fourier Transform (DFT) of the graph. Greedy sampling selection algorithm [8] tries to find near-optimal global solution by following the sequence of locally optimal decisions.

The major drawback of almost all mentioned above graph signal sampling methods is practical impossibility to operate in the big data settings due to their overly extensive computational complexity relying on matrix decompositions, multiplications and other computationally expensive operations. An attempt to address these scalability issues has been made in the paper [3] proposing simple and efficient algorithm based on random walk over the graph. The main idea of the algorithm is to launch random walks starting from arbitrary nodes and terminate them after sufficient number of transitions. The endpoints of these random walks correspond to the nodes which should be added to the sampling set. It is worth mentioning that the algorithm relies only on local graph information, i.e., information about neighboring nodes, and does not use any computationally expensive graph representations and/or mathematical operations. Although the algorithm demonstrates comparable and often better performance than much more complex methods [3, Section IV], in some environments it has been found to oversample large

clusters and undersample small ones [1], which may sometimes lead to the performance degradation. This phenomenon will be discussed in details in the Section 6.1.2.

2.3 Recovering graph signals

In this section we consider how to recover the signal $\mathbf{x}$ based on observing only its samples $\{x[i]\}_{i\in\mathcal{M}}$. In the subsection 2.3.1 we formulate the recovery problem as a convex optimization problem and present two popular methods to solve it, i.e., sparse label propagation (SLP) and network Lasso. In the subsection 2.3.2 we discuss other approaches used for graph signal recovery.

2.3.1 Sparse label propagation and network Lasso

The recovery of the entire graph signal $\mathbf{x}$ from (few) signal samples $\{x[i]\}_{i\in\mathcal{M}}$ is possible for clustered graph signals which do not vary too much over well-connected subsets of nodes (clusters) (cf. [24]). We will quantify how well a graph signal is aligned to the cluster structure using the total variation (TV)

$$\|\mathbf{x}\|_{\mathrm{TV}} := \sum_{\{i,j\}\in\mathcal{E}} W_{ij}|x[j]-x[i]|. \tag{2.1}$$

We learn the graph signal by minimizing the aforementioned total variation (2.1) measure conditioning on known sampling set $\mathcal{M}$:

$$\begin{aligned} &\hat{\mathbf{x}}^{\mathcal{M}} \in \arg\min_{\tilde{\mathbf{x}}} \|\tilde{\mathbf{x}}\|_{\mathrm{TV}} \\ &\text{s.t. } \tilde{x}[i]=x[i] \text{ for all } i\in\mathcal{M}. \end{aligned} \tag{2.2}$$

The problem of the form (2.2) is known as a sparse label propagation and has been extensively studied in [24]. This is a convex optimization problem with non-differentiable objective and therefore precludes using traditional gradient-based algorithms which highly rely on derivatives. However, efficient method for solving (2.2) based on the primal-dual method of Chambolle and Pock [7] has been proposed by Jung et. al. [22]. Moreover, the study has formulated recovery procedure as a message passing, allowing for the distributed implementation which can be especially useful in the big data settings. Sometimes it can be convenient to represent total variation in the form of matrix multiplication rather than the cumulative summation (2.1). In order to do this, we convert our initial undirected graph $\mathcal{G}$ into its directed version $\overrightarrow{\mathcal{G}}$ by setting orientations of edges arbitrarily. Next, we define

incidence matrix $\mathbf{D} \in \mathbb{R}^{|\mathcal{E}|\times|\mathcal{V}|}$ of the following form:

$$D_{ev} := \begin{cases} W_e, v = e^+ \\ -W_e, v = e^- \\ 0, \text{ otherwise} \end{cases} \tag{2.3}$$

It is easy to verify that the total variation of the form (2.1) can be redefined using incidence matrix $\mathbf{D}$ in the following way:

$$\|\mathbf{x}\|_{\text{TV}} := \|\mathbf{D}\mathbf{x}\|_1 \tag{2.4}$$

Consequently, convex optimization problem (2.2) becomes

$$\begin{aligned} &\hat{\mathbf{x}}^{\mathcal{M}} \in \arg\min_{\tilde{\mathbf{x}}} \|\mathbf{D}\mathbf{x}\|_1 \\ &\text{s.t. } \tilde{x}[i] = x[i] \text{ for all } i \in \mathcal{M}. \end{aligned} \tag{2.5}$$

Presented equivalent optimization problem (2.5) can be found particularly useful when is used as an input to disciplined convex optimization solvers, such as CVX [12], which allow using only a limited number of algebraic primitives extensively optimized in terms of performance.

It should be noted that both formulations either (2.2) or (2.5) constitute constrained optimization problems. However, the unconstrained version for solving graph signal recovery can be provided as well in the form of the recently proposed network Lasso method [14, 25] which can be defined by the following convex optimization problem:

$$\hat{\mathbf{x}} \in \arg\min_{\tilde{\mathbf{x}}} \lambda\|\tilde{\mathbf{x}}\|_{\text{TV}} + \sum_{i\in\mathcal{M}} (\tilde{x}[i] - x[i])^2 \tag{2.6}$$

The network Lasso (2.6) conveniently separates convex optimization objective into total variation and empirical error parts. The former is immediately responsible for graph signal recovery itself and is identical to one in the SLP (2.2). The latter term represents an empirical error over the sampling set and penalizes deviations of the signal values being optimized from their true values. Regularization coefficient λ is an essential part of the problem and controls relative preference and impact of the terms on the optimization objective. It also helps to understand the connection between SLP and network Lasso. In particular, when $\lambda \to 0$, the solution of network Lasso is drawn towards the solution of SLP. This happens because the empirical error term gets much higher preference and in order to minimize it optimization variables are forced to the same values as true labels. However, it is important to note that λ must never be 0, because in this case TV term will not

be able to enforce graph signal recovery on the nodes not belonging to the sampling set and graph signal at these nodes may take arbitrary values.

2.3.2 Overview of other recovery methods

Although the SLP and network Lasso described in the previous subsection are quite popular in the GSP world, here we briefly list other methods which can be used for graph signal recovery.

We start with discussing label propagation (LP) [53] which is somewhat similar to SLP presented earlier in this chapter. The key idea of the algorithm is that each node is associated with a soft label represented as a probability distribution. These labels propagate to the neighboring nodes with some probability which is proportional to the edge weights. The effect of such an operation in practice is that the label distributions of the nodes are calculated as weighted averages of the distributions of their neighbors. A fundamentally different approach is described by Romero et. al. [37] and views procedure of learning the labels from the perspective of kernel methods. The idea of this approach reduces to obtaining kernel matrix $\mathbf{K} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ for the graph and then applying conventional kernel algorithms, such as kernel Ridge regression or kernel support vector machines. In their paper authors also describe several ways of obtaining kernel matrix and derive efficient numerical algorithm for solving kernel regression. This kernel-based approach has been recently extended to the Gaussian processes (GP) over graphs presented by Venkitaraman et. al. [46].

Additionally, there is a set of relatively simple and straightforward methods known as average consensus and community-based labelling [32, Chapter 3]. The former method simply assigns to all the nodes such a constant which minimizes least squares error with respect to the signal values in the sampling set. This method does not take into account graph topology or any structural information and, therefore, in many cases demonstrates poor performance, but often can still be used as a baseline. The latter group of methods relies on the prior clustering by applying graph clustering or any other community detection algorithms. Then, all unlabelled nodes in each community are assigned signal values based on the labels of their labelled counterparts.

Chapter 3

Reinforcement learning fundamentals

In this chapter we describe the main concepts and methods for solving RL problems. First, we give definition of the RL, provide a theoretical overview and compare it to the existing machine learning methods. We then briefly introduce Markov decision processes (MDP), the most fundamental reinforcement learning model. Second, we provide the detailed description of a simple but efficient multi-armed bandit (MAB) algorithm which will be further referenced in the subsequent chapters. In Section 3.3 we describe Q-learning which is currently one of the most popular model-free RL method. Finally, the last two sections 3.4 and 3.5 focus on popular deep RL algorithms which apply deep neural networks to solve RL problems.

3.1 Reinforcement learning and Markov decision processes

RL is a subfield of machine learning and artificial intelligence which models the process of learning through the mechanism of benefits (rewards) and punishments. RL specifies behavior of an agent in the environment, and adjusts its actions to maximize the reward. The agent interacts with the environment (see Figure 3.1) by executing actions from a predefined set and observing the response of the environment. At the following stage, the agent is more likely to choose actions which, according to its definition, have been the most successful, i.e., have resulted in the largest reward.

The action-observation procedure repeats for a sufficient number of iterations until the agent learns the policy – set of rules determining what to do in every possible situation.

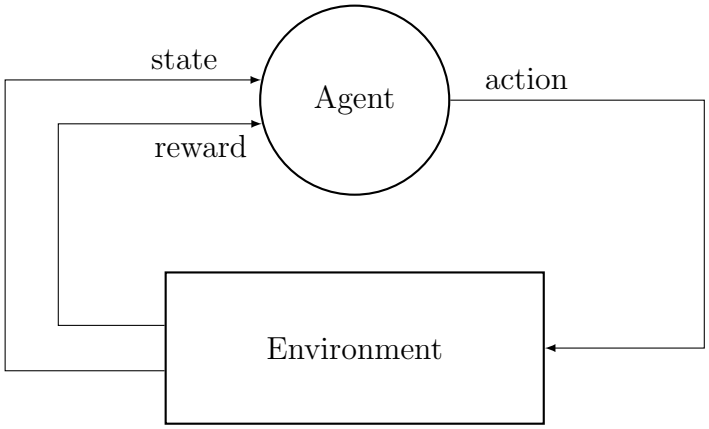


Figure 3.1: Interaction between agent and environment in reinforcement learning.

RL in general can be seen as a larger framework which incorporates different shades of supervised and unsupervised machine learning methods. On one hand, the definition of state/action spaces and reward should be, as a rule, provided by an external supervisor. On the other hand, once these essences are known, the learning agent starts operating fully autonomously and in the unsupervised manner. In addition to this, in many applications individual supervised and unsupervised learning algorithms can be used as building blocks of a larger RL system. For instance, a robot-cleaner can use fully supervised computer vision algorithm for obstacle detection and/or recognition and then use this information as an input of the high level RL system which controls movements of the robot.

Currently, there are two major classes of RL algorithms: model-based and model-free methods. The former can be used when the environment is known, meaning that for each action taken by the agent the probability of being in a certain state is known. The latter do not assume any prior knowledge about the environment and enable learning with trial-and-error from the experience. While model-based methods are usually used in some specific applications with less demands for generalization, model-free ones often attract more attention due to their practicality and applicability to a wider range of real-world problems.

RL algorithms operate in an environment which can be described by the state space $S = \{s_i | i = 1..N\}$ with N states and action space $A = \{a_i | i = 1..M\}$ with M actions. The environment also specifies transition probabilities $P(s, a, s')$ and rewards $R(s, a, s')$ of going from state s to s'

when executing an action a. There is also a discount factor $\gamma \in \{0..1\}$ involved, responsible for diminishing values of rewards obtained in the far future. Based on the notation above, we introduce the simplest model-based RL approach – Markov decision process (MDP). MDP relies on the Bellman equation, estimating a quality function Q of each state-action pair:

$$Q(s,a) = \sum_{s' \in S} P(s,a,s')\big(R(s,a,s') + \gamma Q(s')\big) \tag{3.1}$$

The most popular algorithms for solving MDP and finding optimal policy are value iteration [4] and policy iteration methods [17]. In this thesis we do not stress much attention on them, instead we focus on more relevant to our research model-free methods described in the following sections.

3.2 Multi-armed bandits and bandit optimization

MAB is a problem of selecting the best (in the sense of maximizing reward) alternative among all available ones. The term itself originates from the notion of one-armed bandit which is an informal name of the gambling slot machine with a single lever placed on the side. In one-armed bandit settings a gambler pulls the arm and observes the reward. The reward of a particular bandit slot machine is a random variable with unknown reward distribution.

In contrast to one-armed bandit case, MAB problem is defined over the environment with k one-armed bandits having their own reward distributions (see Figure 3.2). At each time step a gambler faces the decision concerning which arm to pull. The ultimate goal is to maximize the obtained reward after n consecutive arm pulls, but the major problem is that the underlying reward distributions of individual arms are unknown. Despite the fact that the exact reward distribution of i-th arm is unknown, its estimate Q_i can be obtained empirically and the best arm a can be eventually selected by maximizing estimates over arms:

$$Q_i^{(t)} = \frac{\sum_{j=1}^{t-1} R^{(j)}(i)}{N(i)} \tag{3.2}$$

$$a^{(t)} = \arg\max_i Q_i^{(t)}, \tag{3.3}$$

where $N(i)$ denotes number of times i-th arm has been pulled after total $(t-1)$ arm pulls and $R^{(j)}(i)$ denotes the reward obtained on j-th time step

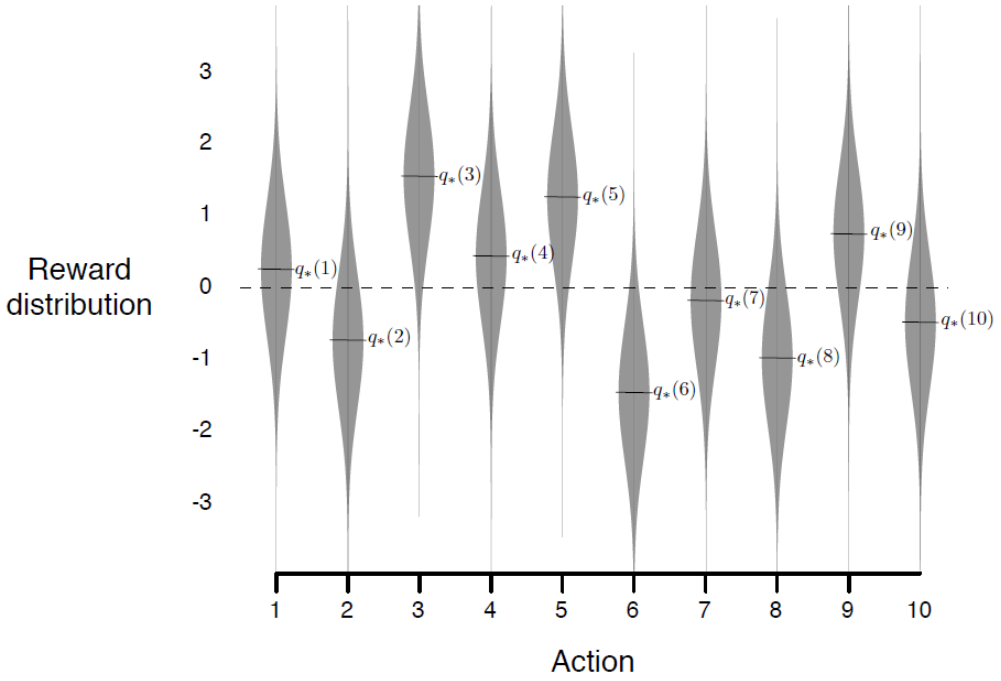


Figure 3.2: Example of the reward distributions of the MAB with 10 arms. q_*(1-10) denote means of the distributions. Image source: [40, Chapter 2, Figure 2.1]

.

for the arm i. If the arm i was not pulled on a certain time step, the reward is 0.

One important point to mention is that the reward estimates may not be totally precise, so that the arm which is the best at the moment may not be the best in the global sense under the sufficiently long run. Therefore, an efficient algorithm should not only exploit currently best arm but also explore other potential opportunities. This common to many RL algorithms problem is often referred to as "exploration-exploitation dilemma".

There are many ways how to handle exploration-exploitation and arm selection at each time step with the most popular ones being ϵ-greedy exploration, upper confidence bound and gradient MAB. We will briefly discuss all the aforementioned approaches in the upcoming subsections.

3.2.1 ϵ-greedy exploration

ϵ-greedy is a simple algorithm (see Algorithm 1) which in addition to selecting currently optimal actions by (3.3) also triggers randomly non-optimal ones

with probability ϵ.

In many cases ϵ is linearly decreased over time so that the operation starts with pure exploration and almost random action selection ($\epsilon \approx 1$) by proceeding with more deterministic actions as long as ϵ is gradually decremented down to the predefined ϵ_{min} value which is usually set relatively small (about 0.05-0.1).

Algorithm 1 Multi-armed bandit with ϵ-greedy exploration

Input: ϵ_{min}, ϵ_{max}, $decaySteps$
Initialize: $Q := \mathbf{0}$, $N := \mathbf{0}$, $R := \mathbf{0}$, $\Delta\epsilon := \frac{\epsilon_{max} - \epsilon_{min}}{decaySteps}$, $\epsilon := \epsilon_{max}$

repeat
  if RAND0TO1() $\leqslant \epsilon$ **then**
    $i :=$ SAMPLERANDOMARM()
  else
    $i := \arg\max_i Q(i)$
  end if
  $r :=$ PULLARM(i)
  $N(i) := N(i) + 1$
  $R(i) := R(i) + r$
  $Q(i) := R(i)/N(i)$
  $\epsilon :=$ MAX($\epsilon_{min}, \epsilon - \Delta\epsilon$)
until terminated

The presented algorithm ensures high exploration rate in the beginning of the learning process, when the uncertainty about the action selection is high, and then slowly converges to the best possible alternative.

3.2.2 Upper confidence bound action selection

Although ϵ-greedy approach performs reasonably well in a great number of situations, it may not be very efficient in terms of convergence speed. The problem is that it enforces exploration randomly, i.e., by triggering random arms, whereas it might be more reasonable to pull arms with the highest uncertainty about the reward. Upper confidence bound (UCB) approach (see Algorithm 2) takes this into account with help of the following update rule:

$$a = \arg\max_i \left(Q_i + c\sqrt{\frac{\ln t}{N(i)}} \right), \tag{3.4}$$

where t – is a time index (or iteration index), $N(i)$ – total number of times the i-th arm has been pulled, c – regularization constant.

The intuition behind (3.4) is that the term $\sqrt{\frac{\ln t}{N(i)}}$ quantifies the uncertainty of the i-th action/arm which depends on the number of times this arm has been pulled. Small number of pulls forces the aforementioned term to become larger which, in turn, encourages selection of such actions.

Algorithm 2 Multi-armed bandit with UCB action selection

Input: c
Initialize: $Q := \mathbf{0}$, $N := \mathbf{0}$, $R := \mathbf{0}$, $t := 0$
1: **repeat**
2: $\quad i := \arg\max_i \left(Q(i) + c\sqrt{\frac{\ln t}{N(i)}} \right)$
3: $\quad r := \textsc{PullArm}(i)$
4: $\quad N(i) := N(i) + 1$
5: $\quad R(i) := R(i) + r$
6: $\quad Q(i) := R(i)/N(i)$
7: $\quad t := t + 1$
8: **until** terminated

3.2.3 Gradient multi-armed bandit with stochastic policy

There is another, probabilistic perspective to finding a solution of the MAB problem. In contrast to the methods discussed in the previous subsections, in the gradient bandit a policy is represented by the probability distribution $\pi^{(\mathbf{w})}$ over the individual arms. In particular, this distribution is parametrized by the weight vector $\mathbf{w} = \{w_1, w_2, ..., w_k\}$ via the softmax rule:

$$\pi^{(\mathbf{w})}(i) = \frac{e^{w_i}}{\sum_{j=1}^{k} e^{w_j}}$$

The arm to be pulled is drawn randomly from this probability distribution. Such an approach ensures exploration automatically, because all actions, including non-optimal ones, have non-zero probabilities and sooner or later will be drawn from the probability distribution. After observing the

reward R_t parameters update can be performed using the stochastic gradient ascent algorithm in the following manner:

$$w_a := \begin{cases} w_a + \alpha R_t(1 - \pi(a)), a = a_t \\ w_a - \alpha R_t \pi(a), \forall a \neq a_t \end{cases}, \tag{3.5}$$

where R_t – reward on the t-th time step, α – learning rate, $\pi(a)$ – probability of pulling arm a, a_t – arm selected on t-th time step, w_a – weight associated with arm a.

The detailed derivation of the (3.5) can be found in the book [40] and is not presented here. The complete algorithm of the gradient MAB can be summarized with the pseudo-code (see Algorithm 3).

Algorithm 3 Multi-armed bandit with stochastic policy

Input: $\mathbf{w}$

1: **repeat**
2: $i := \textsc{SampleAction}(\pi^{(\mathbf{w})})$
3: $r := \textsc{PullArm}(i)$
4: $w_j := \begin{cases} w_j + r(1 - \pi(j)), \text{ if } j = i \\ w_j - r\pi(j), \forall j \neq i \end{cases}$
5: **until** terminated

Another appealing detail in favor of using gradient MAB is that it allows us to specify a prior probability distribution by providing initial weight vector $\mathbf{w}$, which in some situations can significantly speed up the convergence process.

3.3 Q-learning

Q-learning is a popular model-free RL method which can operate on the discrete state and action spaces in the environments with priorly unknown transition probabilities and rewards [48]. The method amounts to learning Q-function which is usually represented as a two-dimensional table with rows being states and columns being actions. During the learning phase rewards from the terminal states backpropagate to the other states with help of the following update equation:

$$Q(s, a) = Q(s, a) + \alpha(r + \gamma \max_a Q(s', a)), \tag{3.6}$$

where $\alpha \in \{0..1\}$ – learning rate, $\gamma \in \{0..1\}$ – discount factor, $Q(s,a)$ – action-value function for the state-action pair (s,a), $Q(s',a)$ – action-value function for the state-action pair (s',a), r – immediate reward.

In order for Q-learning to converge it is necessary that each state is visited sufficient number of times to get accurate estimate of the Q-function in this state. In practice this is only feasible when the state space is small.

Q-learning is known as off-policy method, meaning that it uses two different policies: behavioral policy for learning Q-table and optimal policy for operation. Behavioral policy is chosen so that it ensures sufficient amount of exploration by taking many potentially suboptimal but exploratative actions, whereas optimal policy exploits actions which maximize the reward. This can be opposed to the on-policy methods which use the same policy for estimating Q-values and for operation. In practical applications Q-learning often uses ϵ-greedy policy with ϵ being decreased from 1 to 0 over the sufficient number of episodes.

3.4 Deep Q-network

Q-learning is very efficient in simple environments with small state and action spaces but is not suitable for many real world problems with large state spaces (order of thousand and more). For example, if we consider training a RL agent to play famous Atari games and define a state as a single raw pixel map with 200x200=40000 pixels, then given the number of colors is 256 (for grayscale image), number of states in our state space will be equal to 256^{40000}. Such a large state space precludes the use of table-based methods. Furthermore, since for convergence it is required to visit each state-action pair sufficiently many times, it makes such problems computationally intractable because number of episodes required for converges becomes extremely large. In order to deal with these serious issues it was proposed [33] to replace conventional Q-table with neural network or Q-network, a function approximator which inputs state representation as a multidimensional feature vector and outputs Q-values of the different actions for this state (see Figure 3.3).

Training Q-network comes down to minimizing the following loss function:

$$\mathcal{L} = (r + \gamma \max_a Q(s',a) - Q(s,a))^2, \tag{3.7}$$

where $\mathcal{L}$ – loss function being minimized, r – immediate reward, $Q(s,a)$ – action-value function for the state-action pair (s,a), $Q(s',a)$ – action-value function for the state-action pair (s',a), γ – discount factor.

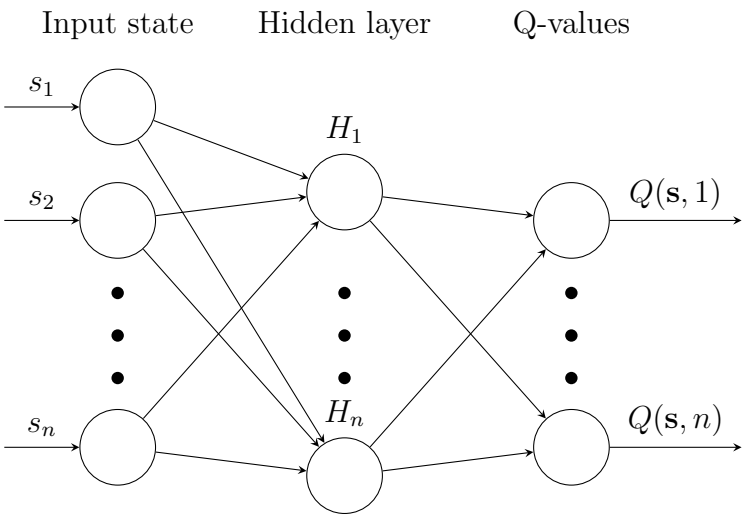


Figure 3.3: Deep Q-network architecture.

Intuition behind the equation (3.7) is that two adjacent states s' and s only differ with respect to the discount factor γ and transition (immediate) reward r. Therefore, difference between values of adjacent states should be as small as possible, which is achieved by minimizing the function specified above.

In order to improve the convergence speed and stability of Q-network algorithm, additional measures should also be taken. This includes using experienced replay and freezing target network techniques [33].

Experienced replay is aimed at breaking correlations between adjacent states during the training process and preventing the optimization procedure being trapped into a poor local minimum of the objective function (3.7). The main idea is to store all transitions between states into replay memory in the form of tuple (s_t, a, s_{t+1}, r) and use them later during the training phase. Replay memory is usually large enough to retain several hundred thousands transitions from few hundreds previous episodes. During the training phase, instead of using the most recent transitions, we randomly sample minibatch of transitions from the replay memory and use them for fitting neural network.

Target network freezing implies that the computation of $\max_a Q(s', a)$ in (3.7) should be done using separate neural network ("target network") which is periodically (every several hundred episodes) synchronized with the main neural network whose weights are being constantly adjusted on each iteration. This allows to stabilize computation of the target values $\max_a Q(s', a)$.

Algorithm 4 Q-network algorithm with experienced replay and target network freezing

Input: initial state s, memory $\mathcal{M}$, Q_{main}, Q_{target}, ϵ_{min}, ϵ_{max}, $\Delta\epsilon$, updateInterval

Initialize: ep := 0, $\epsilon := \epsilon_{max}$

repeat
  if RAND0TO1() $< \max(\epsilon, \epsilon_{min})$ **then**
    a := GETRANDOMACTION(s)
  else
    $a := \arg\max_i Q_{main}(s, a_i)$
  end if
  s_{next}, r := GETNEXTSTATE(s)
  add (s, s_{next}, r, a) to replay memory $\mathcal{M}$
  sample minibatch $(s^{(i)}, s^{(i)}_{next}, r^{(i)}, a^{(i)})$ from memory $\mathcal{M}$
  $y^{(i)} := \begin{cases} r^{(i)}, \text{if } s^{(i)} \text{ is terminal state} \\ r^{(i)} + \gamma \max_a Q_{target}(s^{(i)}_{next}, a^{(i)}), \text{otherwise} \end{cases}$
  train Q_{main} using $s^{(i)}$ as inputs and $y^{(i)}$ as responses
  if ep % updateInterval == 0 **then**
    $Q_{target} := Q_{main}$
  end if
  ϵ := MAX($\epsilon_{min}, \epsilon - \Delta\epsilon$)
  ep := ep + 1
  $s := s_{next}$
until terminated

The classical Q-network algorithm (see Algorithm 4) had been shown to have several drawbacks which were partially eliminated in the more recent studies. In particular, due to the **argmax** operator used in action selection and target values evaluation, the algorithm tends to deliver too optimistic estimates of Q-values, which can often lead to suboptimal policies. The solution to this problem is proposed in the study [45] which introduces the idea of Double Deep Q-Network (DDQN). DDQN architecture decouples target values computation and the best action selection operations by using two different Q-networks which are updated interchangeably. Another extension is discussed in the research paper [47] and revolves around Dueling Deep Q-Network (Dueling DQN). The core idea of this concept is to split hidden layer into two parts (streams) computing estimates of state values and actions advantages independently. These estimates are then combined by the output layer to produce familiar Q-values. It is interesting to mention that

several advanced Q-network methods, including ones described in the current section, can be combined into Rainbow algorithm [16] which has been claimed to achieve unprecedented convergence speed and performance in the Atari environment.

3.5 Policy gradient

In contrast to Q-network algorithm policy gradient (see Algorithm 5) is not used to approximate Q-table, instead, it learns target policy directly by mapping input state to the probability distribution of actions (see Figure 3.4).

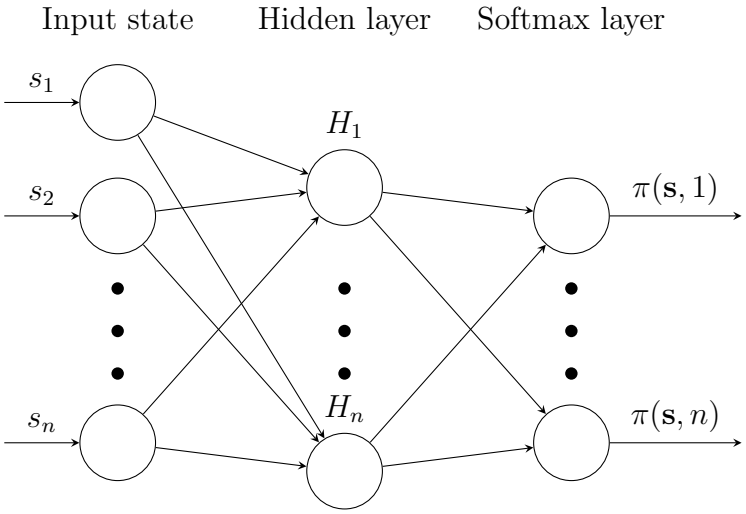


Figure 3.4: Policy network architecture.

According to the policy gradient theorem [41], gradient of any differentiable policy can be expressed as

$$\nabla \mathbf{w} = \mathbb{E}_\pi\{\nabla \log \pi^{(\mathbf{w})}(s, a) Q(s, a)\}, \tag{3.8}$$

where $\nabla \mathbf{w}$ – gradient of weights of the policy network, $\pi^{(\mathbf{w})}(s, a)$ – policy network which outputs probability of the action a in the state s, $Q(s, a)$ – action-value function for the state-action pair (s, a), $\mathbb{E}_\pi\{\cdot\}$– expectation under the policy $\pi^{(\mathbf{w})}$.

The closed-form gradient expression (3.8) allows using gradient-based optimization methods for finding the optimal policy.

In many RL tasks (e.g. Go game) reward only comes upon reaching the terminal state bringing us to the so-called credit assignment problem, one

Algorithm 5 Policy gradient algorithm

Input: initial state s, policy network $\pi^{(\mathbf{w})}$, buffer $\mathcal{M}$
  repeat
    repeat
      $a :=$ SampleAction$(\pi^{\mathbf{w}}(s))$
      s_{next}, $r :=$ GetNextState(s, a)
      add (s, a, r) to the buffer $\mathcal{M}$
      $s := s_{next}$
    until episode has finished
    (optional) apply variance reduction to the rewards in $\mathcal{M}$
    for `(`s, a, r`) in` $\mathcal{M}$ **do**
      $\mathbf{w} := \mathbf{w} + \nabla \log \pi^{(\mathbf{w})}(s, a)r$
    end for
  until terminated

of the most fundamental problems in the RL field. This problem reduces to the fact that it is difficult to determine which actions (and to which extent) taken during the episode contributed to achieving this specific reward. Consequently, it becomes non-trivial to estimate action-value function $Q(s, a)$ of the state-action pairs visited during the episode. There are at least two ways how to handle this issue. The first and the most naive approach is to treat the finally obtained reward as a sample of the action-value function for all state-action pairs which were observed during the episode. Then for each state-action pair these samples can be averaged over sufficient number of episodes to produce estimates of the action-values function. Practically this often works well, but the convergence speed may decrease dramatically and the finally obtained policy may be suboptimal. The second approach is to try to estimate values $Q(s, a)$ of state-action pairs using separate neural network $Q^{\theta}(s, a)$ in parallel with learning action gradients. These estimates $Q^{\theta}(s, a)$ will then replace the action-value function $Q(s, a)$ in the gradients equation (3.8):

$$\nabla \mathbf{w} = \mathbb{E}_{\pi}\{\nabla \log \pi^{(\mathbf{w})}(s, a) Q^{\theta}(s, a)\}, \tag{3.9}$$

where $\nabla \mathbf{w}$ – gradient of weights of the policy network, $\pi^{(\mathbf{w})}(s, a)$ – policy network which outputs probability of action a in the state s, $Q^{\theta}(s, a)$ – neural network with weights θ estimating action-value function for the state-action pair (s, a), $\mathbb{E}_{\pi}\{\cdot\}$– expectation under the policy $\pi^{(\mathbf{w})}$.

This is commonly known as actor-critic approach and is specifically meant

for dealing with credit assignment problem and improving learning performance of the policy gradient. In this case, the actor is a conventional policy gradient performing action selection, whereas the critic is a standard DQN algorithm estimating action-value function $Q(s, a)$ for each state-action pair.

It is worth mentioning that in addition to vanilla policy gradient and actor-critic approach there are several other policy gradient variations fundamentally different from the classical versions. For example, the study [39] introduces and proves convergence of the deterministic policy gradient which outputs exact action rather than a probability distribution over actions. The study [28] extends this idea to continuous action spaces, which allows using policy gradient in continuous control algorithms. Natural policy gradient [26] is another perspective research direction which takes different view on the gradient computation itself and is claimed to provide promising results.

Chapter 4

Graph signal sampling as reinforcement learning problem

In the previous chapters we considered theoretical foundations of machine learning for graph signal recovery as well as RL basics. In this chapter we turn our theoretical knowledge into practice by developing novel algorithms for graph signal sampling. We start our discussion by proposing MAB-based RL algorithm in the Section 4.1 followed by the deep RL algorithm utilizing policy gradient approach in the Section 4.2. Finally, Section 4.3 discusses difficulties associated with usage of RL for graph signal recovery.

4.1 Proposed sampling algorithm based on the stochastic MAB

The problem of selecting the sampling set $\mathcal{M}$ and recovering the entire graph signal $\mathbf{x}$ from the signal values $x[i]$ can be interpreted as a RL problem. Indeed, we consider the selection of the nodes to be sampled being carried out by an "agent" which crawls over the empirical graph $\mathcal{G}$. The set of actions our sampling agent may take is $\mathcal{A} = \{1, \ldots, H\}$.

A specific action $a \in \mathcal{A}$ refers to the number of hops the sampling agent performs starting at the current node i_t to reach a new node i_{t+1}, which will be added to the sampling set, i.e., $\mathcal{M} := \mathcal{M} \cup \{i_{t+1}\}$. In particular, the new node i_{t+1} is selected uniformly at random among the nodes which belong to its a-step neighbourhood $\mathcal{N}(i_t, a)$ (see Figure 4.1).

The problem of optimally selecting actions at given time can be formulated as a MAB problem. Each arm of the bandit is associated with an action. In our setup, a sampling strategy (or policy) amounts to specifying a probability distribution over the individual actions $a \in \mathcal{A}$. We parametrize

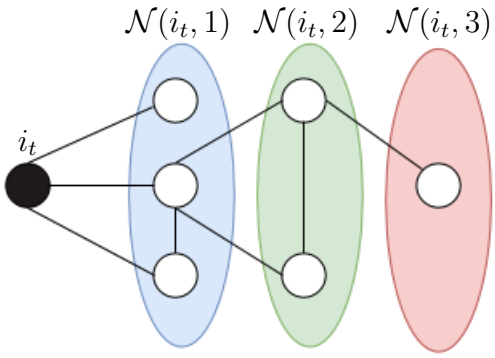


Figure 4.1: The filled node represents the current location i_t of the sampling agent at time t. We also indicate the 1-, 2- and 3-step neighbourhoods.

this probability distribution with a weight vector $\mathbf{w} = (w_1, \ldots, w_H) \in \mathbb{R}^H$ using the softmax rule:

$$\pi^{(\mathbf{w})}(a) = \frac{e^{w_a}}{\sum\limits_{b \in \mathcal{A}} e^{w_b}}$$

The weight vector $\mathbf{w}$ is tuned in the episodic manner with each episode amounting to selecting sampling set $\mathcal{M}$ based on the policy $\pi^{(\mathbf{w})}$. At each timestep t the agent randomly draws an action a_t according to the distribution $\pi^{(\mathbf{w})}$ and performs transition to the next node i_{t+1} which is selected uniformly at random from the a_t-step neighbourhood $\mathcal{N}(i_t, a_t)$. As was mentioned earlier, the node t_{t+1} is added to the sampling set, i.e., $\mathcal{M} := \mathcal{M} \cup \{i_{t+1}\}$. We also record the action a_t and add it to the action list, i.e., $\mathcal{L} := \mathcal{L} \cup \{a_t\}$. The process continues until we obtain a sampling set $\mathcal{M}$ with the prescribed size (sampling budget) M.

Our goal is to learn an optimal policy $\pi^{(\mathbf{w})}$ for the sampling agent in order to obtain signal samples which allow recovery of the entire graph signal with minimum error. We assess the quality of the policy using the MSE incurred by the recovered signal $\hat{\mathbf{x}}^{\mathcal{M}}$ which is obtained via (2.2) using the sampling set $\mathcal{M}$ by following policy $\pi^{(\mathbf{w})}$:

$$R := -\frac{1}{N} \sum_{j \in \mathcal{V}} (x[j] - \hat{x}^{\mathcal{M}}[j])^2$$

The obtained reward is associated with all actions/arms which contributed to picking samples into sampling set during the episode. For example, if the sampling set has been obtained by pulling arms 1, 2 and 5, the obtained reward will be associated with all these arms, because we do not know what is the exact contribution of the specific arm to the finally obtained MSE.

The key idea behind gradient MAB is to update weights $\mathbf{w}$ so that actions yielding higher rewards become more probable under $\pi^{(\mathbf{w})}$ [40, Chapter 2.8]. According to the aforementioned book weights update can be accomplished using gradient ascent algorithm:

$$w_a := \begin{cases} w_a + \alpha R(1 - \pi(a)), a = a_k \\ w_a - \alpha R \pi(a), \forall a \neq a_k \end{cases} \\ \text{for } k = 1..M-1, a \in \mathcal{A}, a_k \in \mathcal{L} \tag{4.1}$$

The single difference between update rule (4.1) and one presented in the book [40, Eq. 2.10] is that in our case weights update is performed in the end of each episode and not after an arm pull. That is because we do not know reward immediately after pulling an arm and should wait until the whole sampling set is collected and reward is observed. The intuition behind the update equation (4.1) is that for each arm which has participated in picking a node into sampling set ($a = a_t$), the weight is increased, whereas weights of remaining arms ($\forall a \neq a_t$) are decreased. In both cases degree of weight increase/decrease is scaled by the reward obtained with help of this arm as well as by the learning rate α. For faster convergence in our implementation, instead of stochastic gradient ascent we use mini-batch gradient ascent in combination with RMSprop technique [44] (see Algorithm 6 for implementation details).

Choice of the gradient MAB algorithm can be additionally justified by the study [5] which shows that in the environments with non-stationary rewards probabilistic MAB policy can result in higher expected reward in comparison to single-best action policies. In our problem non-stationarity of reward arises from the graph structure itself, i.e., reward distribution for a particular arm of a bandit depends on the location of the sampling agent. Suppose sampling budget M is 2 and consider example presented in Figure 4.2. In case (a) sampling agent is initially located at node 4. By pulling arm #1 it can only pick node 3 which is in the other cluster. It is easy to verify that by using recovery method (2.2) graph signal will be perfectly reconstructed (MSE = 0). On the other hand, case (b) shows the situation when the agent can only pick nodes 2 or 3 belonging to the same cluster as currently sampled node, leading to non-zero reconstruction MSE.

The whole process of weight updates is repeated for sufficient number of episodes until convergence is reached and the optimal stochastic policy is attained. Described above learning procedure can be efficiently summarized in the form of pseudocode (see Algorithm 6).

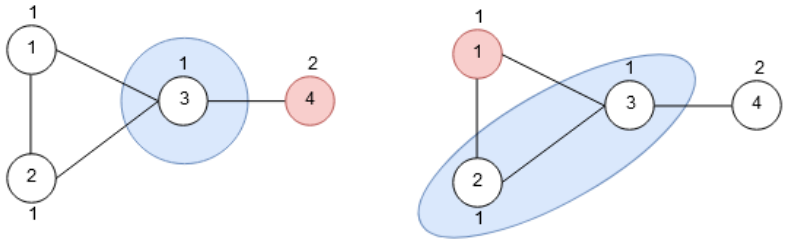
Obtained probability distribution $\pi^{(\mathbf{w})}$ represents sampling strategy which incurs the minimum reconstruction MSE when using the convex recovery

Algorithm 6 Online sampling and reconstruction with MAB

Input: empirical graph $\mathcal{G}$, sampling budget M, batch size B, α
Initialize: $\mathbf{w} := \mathbf{0}, \mathcal{M} = \{\emptyset\}, \mathcal{L} = \{\emptyset\}, \nabla\mathbf{w} = \mathbf{0}, \mathbf{g} = \mathbf{0}, ep = 0$
  repeat
    select starting node $i \in \mathcal{V}$ randomly
    $\mathcal{M} := \{i\}$
    $\mathcal{L} = \{\emptyset\}$
    for $t := 1; t < M$ **do**
      $a :=$ SAMPLEACTION$(\pi^{(\mathbf{w})})$
      $i_{next} :=$ SAMPLENODE$(\mathcal{G}, \mathcal{N}(i, a))$
      $\mathcal{M} := \mathcal{M} \cup \{i_{next}\}$
      $\mathcal{L} :=$ APPENDTOLIST$(\mathcal{L}, a)$
      $i := i_{next}$
    end for
    $\hat{\mathbf{x}} \in \arg\min_{\tilde{\mathbf{x}}} \|\tilde{\mathbf{x}}\|_{\mathrm{TV}}$
    s.t. $\tilde{x}[i] = x[i]$ for all $i \in \mathcal{M}$
    $R := -\frac{1}{N} \sum_{j \in \mathcal{V}} (x[j] - \hat{x}[j])^2$
    for $k := 1; k < M$ **do**
      for $a := 1; a \leqslant H$ **do**

$$\nabla w_a := \begin{cases} \nabla w_a + R(1 - \pi(a)), \text{ if } a = \mathcal{L}[k] \\ \nabla w_a - R\pi(a), \text{ otherwise} \end{cases}$$

      end for
    end for
    $ep := ep + 1$
    if $ep \bmod B = 0$ **then**
      $\mathbf{g} := 0.9\mathbf{g} + 0.1(\nabla\mathbf{w})^2$
      $\mathbf{w} := \mathbf{w} + \alpha\nabla\mathbf{w}/\sqrt{\mathbf{g}}$
      $\nabla\mathbf{w} := \mathbf{0}$
    end if
  until convergence is reached
Output: $\pi^{(\mathbf{w})}$



(a) (b)

Figure 4.2: Illustration of reward being conditioned on the position of a sampling agent. In this picture: red node – current position of the sampling agent, blue region – nodes within distance 1 from the sampling agent. Node indices are shown inside the nodes, signal values – outside.

method (2.2).

4.2 Sampling algorithm based on the policy gradient

In the previous section we presented an algorithm which did not use any notion of state during the learning process. However, this important information may potentially vastly improve performance of the learning algorithm if a state is chosen appropriately. For instance, at each time step sampling agent may not simply draw an action from the probability distribution, but also take into account state of the current node which can be a compact feature vector of the node in some way summarizing its place within the graph. The intuition is that the nodes belonging to the same clusters should have similar states which, in turn, should trigger similar actions. On the other hand, the problem of choosing suitable state representation for a node is rather challenging. At the first glance, the most straightforward measure which could be used for defining a state is a local clustering coefficient of a node:

$$c_i = \frac{2|\mathcal{E}(\mathcal{N}(i))|}{k_i(k_i - 1)},$$

where $\mathcal{E}(\mathcal{N}(i))$ denotes the set of edges connecting two different neighbors of node i and k_i denotes number of neighbors node i has.

Although the measure summaries relationships between neighbors around current node, it is too local to differentiate between different locations of

nodes in the graph.

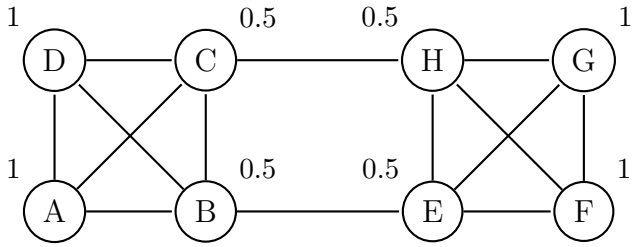


Figure 4.3: Using local clustering coefficient as a state (clustering coefficient values are shown outside the nodes).

From Figure 4.3 it can be easily noticed that multiple nodes, which are in different clusters, may have the same states/clustering coefficient values (e.g. nodes A and F both have clustering coefficient 1). This makes action selection ambiguous and has a detrimental effect on learning process.

This problem can be overcome if instead of clustering coefficients one would use feature vectors, such as node2vec [13]. Node2vec is an embedding algorithm which utilizes popular in natural language processing word2vec paradigm [11] and matches a node to a feature vector of a predefined length. Consequently, this feature vector can be directly used as an input of the policy gradient algorithm (see Algorithm 7).

It should be noted that the feature selection for graph nodes is a non-trivial process and requires careful consideration. The major obstacle is that the derivation of the feature vectors from the data often requires computationally expensive pre-processing making this approach simply impractical.

4.3 Reward choice problem

Traditionally, reward function occupies central part of any reinforcement learning algorithm and has a direct impact on its convergence and general stability. Usually, in RL applications reward serves as a measure of success and is used to encourage actions yielding higher expected reward. For instance, if the goal of an agent is to escape a labyrinth, then the huge reward should be given when the agent accomplishes this task. Similarly, in graph signal sampling the most naive and straightforward measure of success is mean squared error of recovery. Since we aim at achieving the smallest possible reconstruction error, the smaller MSE should lead to the larger reward. This can be accomplished by using negative MSE reward measure, as it was

Algorithm 7 Online sampling and reconstruction with policy gradient

Input: empirical graph $\mathcal{G}$, sampling budget M, batch size B
Initialize: $\mathbf{w}, \mathcal{M} := \{\emptyset\}, \mathcal{H} := \{\emptyset\}, ep := 0$
  repeat
    select starting node $i \in \mathcal{V}$ randomly
    $\mathcal{M} := \{i\}$
    for $t := 1; t < M$ **do**
      $s :=$ GetState(i)
      $a :=$ SampleAction$(\pi^{(\mathbf{w})}(s))$
      $i_{next} :=$ SampleNode$(\mathcal{G}, \mathcal{N}(i, a))$
      $\mathcal{M} := \mathcal{M} \cup \{i_{next}\}$
      add (s,a,reward placeholder) to history buffer $\mathcal{H}$
      $i := i_{next}$
    end for
    $\hat{\mathbf{x}} \in \arg\min_{\tilde{\mathbf{x}}} \|\tilde{\mathbf{x}}\|_{\mathrm{TV}}$
    s.t. $\tilde{x}[i] = x[i]$ for all $i \in \mathcal{M}$
    $R := -\frac{1}{N} \sum_{j \in \mathcal{V}} (x[j] - \hat{x}[j])^2$
    populate reward placeholders in $\mathcal{H}$ with actual reward R
    $ep := ep + 1$
    if $ep \bmod B = 0$ **then**
      for each (s,a,r) in $\mathcal{H}$ **do**
        fit policy network $\pi^{(\mathbf{w})}$ using (s,a,r)
      end for
      clear $\mathcal{H}$
    end if
  until convergence is reached
Output: $\pi^{(\mathbf{w})}$

already demonstrated in the previous sections:

$$R := -\frac{1}{N}\sum_{j\in\mathcal{V}}(x[j]-\hat{x}^{\mathcal{M}}[j])^2$$

The problem with using the aforementioned reward is that for its calculation one should know beforehand true underlying graph signal. This measure is acceptable in such applications as data compression where we already have all the data at our disposal and need to select sampling set which would minimize decoding error. However, in problems related to regression or data reconstruction from only finite number of known measurements usage of such a reward function is problematic. One possible reward measure which does not rely on true signal values and is available immediately from the graph signal recovery formulation is the objective function of the SLP algorithm:

$$\begin{aligned} R := &-\min_{\tilde{\mathbf{x}}} \|\tilde{\mathbf{x}}\|_{\mathrm{TV}} \\ &\text{s.t. } \tilde{x}[i]=x[i] \text{ for all } i\in\mathcal{M}. \end{aligned} \tag{4.2}$$

However, it is disputable what is the relation between optimal value of the SLP and the quality of the recovered graph signal. It may seem natural to assume that since in the SLP we aim at minimizing total variation, we can use negative SLP objective value as a reward (4.2). This assumption may be misleading, because if obtained signal is smooth it may still have huge mean squared error.

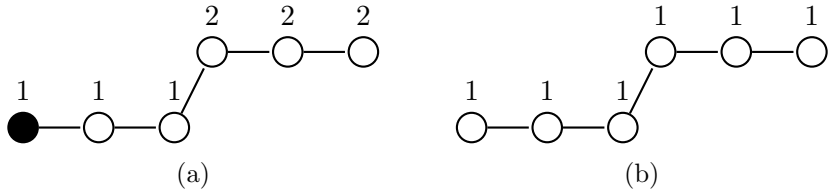


Figure 4.4: Illustration of clustered graph signal and the result of incorrect recovery (a) – underlying true graph signal and sampled node (shown in black), (b) – recovered graph signal.

Consider the graph signal with two clusters depicted in Figure 4.4. Signal values within each cluster are the same and equal to 1 and 2 respectively. Suppose we sample arbitrary node in the first cluster and try to recover whole graph signal based on it. It is easy to verify that by using familiar SLP procedure one can get perfectly smooth signal with optimization objective

being zero. However, in this case all signal values in the second cluster will be completely incorrect and mean squared error will be non-zero.

In addition to the reward measures described above, another measure that may seem reasonable is Shannon's entropy. If we define minimum and maximum values of our graph signal as m and M and quantize the signal to a grid with n divisions then the Shannon's entropy can be calculated according to the following formula:

$$H = -\sum_{i=1}^{n} P_i \log(P_i), \tag{4.3}$$

where P_i is a probability of graph signal values falling into the interval:

$$P_i = Prob\big(\hat{\mathbf{x}}^{\mathcal{M}} \in [m + \frac{(M-m)(i-1)}{n}; m + \frac{(M-m)i}{n})\big). \tag{4.4}$$

Then the reward simply becomes:

$$R := H \tag{4.5}$$

The intuition behind the entropy maximization is that we aim at sampling graph signal in such a way so that the recovered signal carries maximum information or, in the other words, has maximum entropy.

In the Section 5.4 we conduct experiments on the synthetic dataset to understand how Shannon's entropy as well as objective function of SLP are suitable for being used as reward measures and how they are connected to the MSE of signal recovery.

Chapter 5

Numerical experiments

This chapter describes models used in our simulations, experiments performed and results obtained.

5.1 Simulation model

In our experiments we use a synthetic dataset generated in accordance to the stochastic block model (SBM). SBM is a simple probabilistic model for representing graphs which contain communities. Within SBM a graph is partitioned into l clusters with sizes N_l. The probability of having an edge between nodes of the same cluster is denoted p, whereas the probability of edge between the nodes of different clusters is denoted q. Such a model has been found a particularly useful because it has been extensively studied from many perspectives [10, 15, 34] and allows for a relatively simple theoretical analysis. In addition to the SBM model assumption we make sizes of clusters N_l follow the geometric distribution:

$$N_l \sim Geometric(s) \tag{5.1}$$

with probability mass function being

$$Pr(N_l = k) = (1-s)^{k-1}s, k = \{1, 2, 3, ...\} \tag{5.2}$$

The parameter s is known as "probability of success" and ranges from 0 to 1. Formula (5.1) allows generation of clusters with non-uniform sizes (more smaller clusters and fewer larger ones), which enables more consistent imitation of the real-world data.

A simple but useful model for clustered graph signals is:

$$\mathbf{x} = \sum_{\mathcal{C} \in \mathcal{F}} a_{\mathcal{C}} \mathbf{t}_{\mathcal{C}}, \tag{5.3}$$

with the cluster indicator signals

$$t_{\mathcal{C}}[i] = \begin{cases} 1, \text{ if } i \in \mathcal{C} \\ 0 \text{ else.} \end{cases}$$

The partition $\mathcal{F}$ underlying the signal model (5.3) can be chosen arbitrarily in principle. However, our methods are expected to be the most useful if the partition matches the intrinsic cluster structure of the empirical graph $\mathcal{G}$. The clustered graph signals of the form (5.3) conform with the network topology, in the sense of having small TV $\|\mathbf{x}\|_{\mathrm{TV}}$, if the underlying partition $\mathcal{F} = \{\mathcal{C}_1, \ldots, \mathcal{C}_{|\mathcal{F}|}\}$ consists of disjoint clusters $\mathcal{C}_l$ with small cut-sizes. Relying on the clustered signal model (5.3), [23, Thm. 3] presents a sufficient condition on the choice of sampling set such that the solution $\hat{\mathbf{x}}$ of (2.2) coincides with the true underlying clustered graph signal of the form (5.3). The condition presented in [23, Thm. 3] suggests choosing the nodes in the sampling set preferably near the boundaries between the different clusters.

5.2 Experiments with MAB-based sampling algorithm

We now verify the effectiveness of the proposed sampling set selection algorithm (Algorithm 6) using synthetic data and compare it to two other existing approaches, i.e., random walk sampling (RWS) and URS discussed earlier in Section 2.2. We define a random graph with 10 clusters where sizes of clusters are drawn from the geometric distribution with probability of success $s = 8/100$. In accordance to the SBM intra- and inter-cluster connection probabilities are parametrized as $p = 7/10$ and $q = 1/100$. We then generate a clustered graph signal according to (5.3) with the signal coefficients $a_{\mathcal{C}_l} = l$ for $l = 1, 2, ..., 10$. Example of a typical instance of random graph with such parameters is shown in Figure 5.1.

Given the model we generate training data consisting of $K = 500$ random graphs and for each graph instance we run Algorithm 6 for 10000 episodes, which is sufficient to reach convergence.

In Figure 5.3 we illustrate the mean policy

$$\pi^{(\mathbf{w})} = \frac{1}{K} \sum_{i=1}^{K} \pi_i^{(\mathbf{w})} \tag{5.4}$$

The finally obtained policy (5.4) is then evaluated by applying it to 500 new i.i.d. realizations of the empirical graph, yielding the sampling sets $\mathcal{M}^{(i)}$,

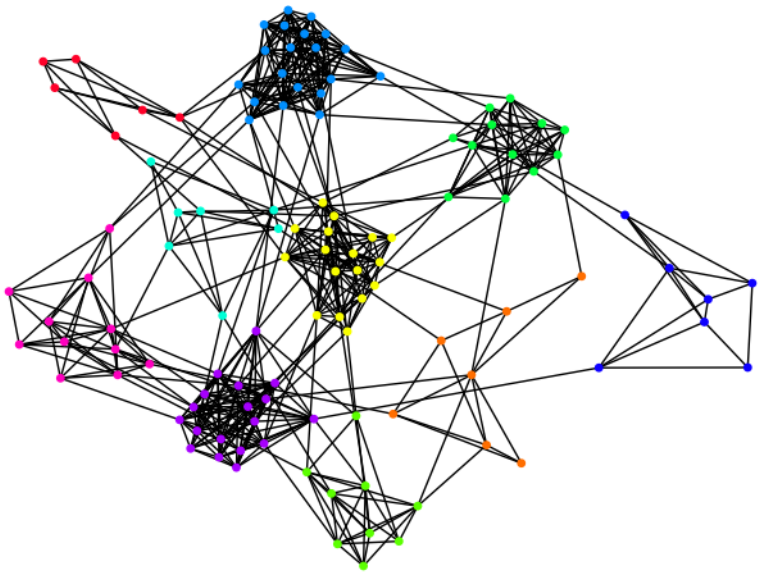


Figure 5.1: Empirical graph obtained from the stochastic block model with $p = 7/10$ and $q = 1/100$.

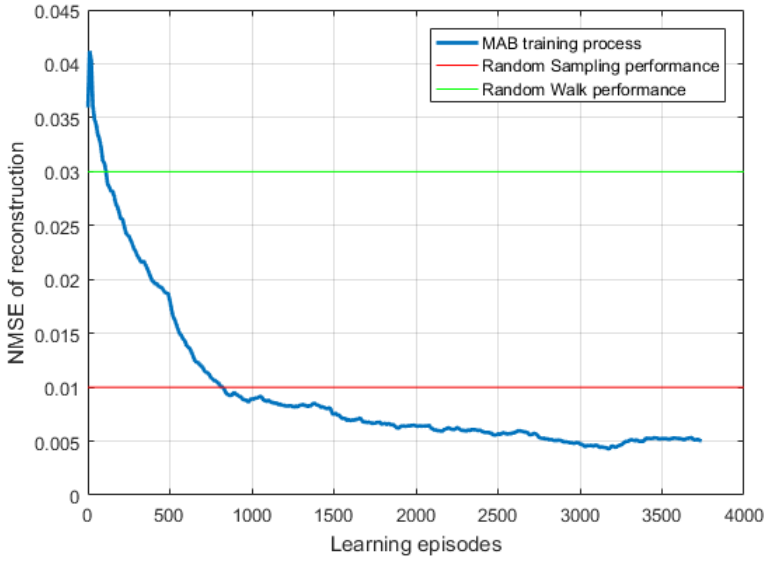


Figure 5.2: Convergence of gradient MAB for one learning instance $\mathcal{G}_i$ (showing first 3700 episodes).

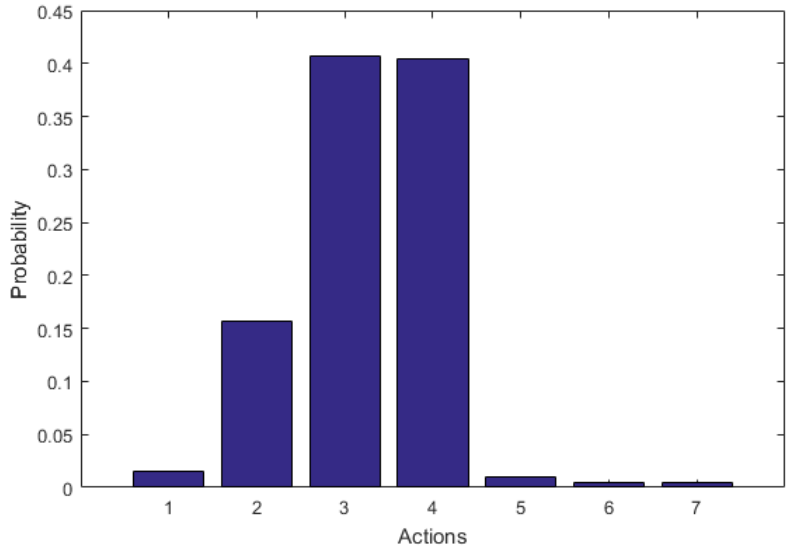


Figure 5.3: Mean policy for the stochastic block model family $\mathcal{G}$.

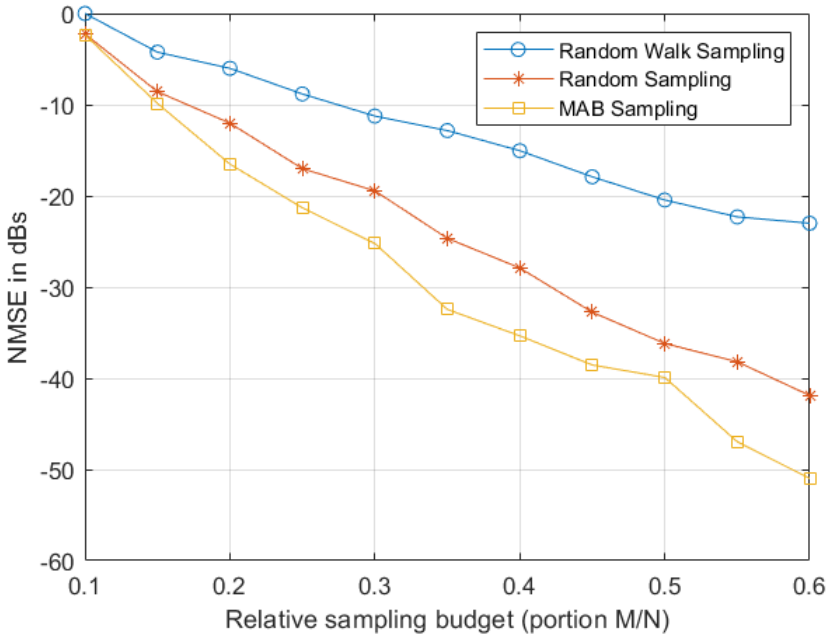


Figure 5.4: Test set error obtained from graph signal recovery based on different sampling strategies.

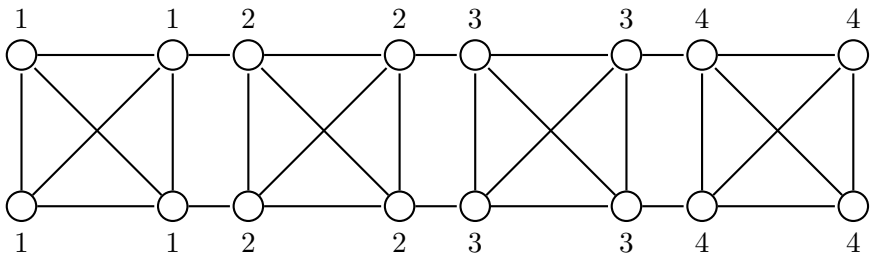


Figure 5.5: Toy model for deep reinforcement learning experiment. Signal values are shown outside of the nodes.

$i = 1, \ldots, 500$, and measuring the normalized mean squared error (NMSE) incurred by graph signal recovery from those sampling sets:

$$NMSE_{\mathcal{G}_i} = \frac{\|\hat{\mathbf{x}}^{(i)} - \mathbf{x}^{(i)}\|_2^2}{\|\mathbf{x}^{(i)}\|_2^2}$$

$$NMSE = \frac{1}{500} \sum_{i=1}^{500} NMSE_{\mathcal{G}_i}$$

We perform similar measurements of the NMSE for random walk and random sampling algorithms under different sampling budgets and convert results to the logarithmic scale (see Figure 5.4).

5.3 Experiments with deep reinforcement learning

In the Section 4.2 we discussed how policy gradient can be potentially used for graph signal sampling. In this section we generate toy graph (see Figure 5.5) which we use to compare stateful policy gradient sampling against stateless MAB sampling. Moreover, in order to study better the impact of the state space choice on the sampling performance we use two different state representations – feature vector obtained with help of node2vec and local clustering coefficient of a node. As a policy network we use a neural network with one hidden layer and 100 neurons in it. Size of the input layer and, respectively, dimensionality of feature vectors in node2vec algorithm is set to 3. Simulation results obtained over 5000 episodes are shown in Figure 5.6. We also illustrate layout of the features in the node2vec feature space in the Figure 5.7.

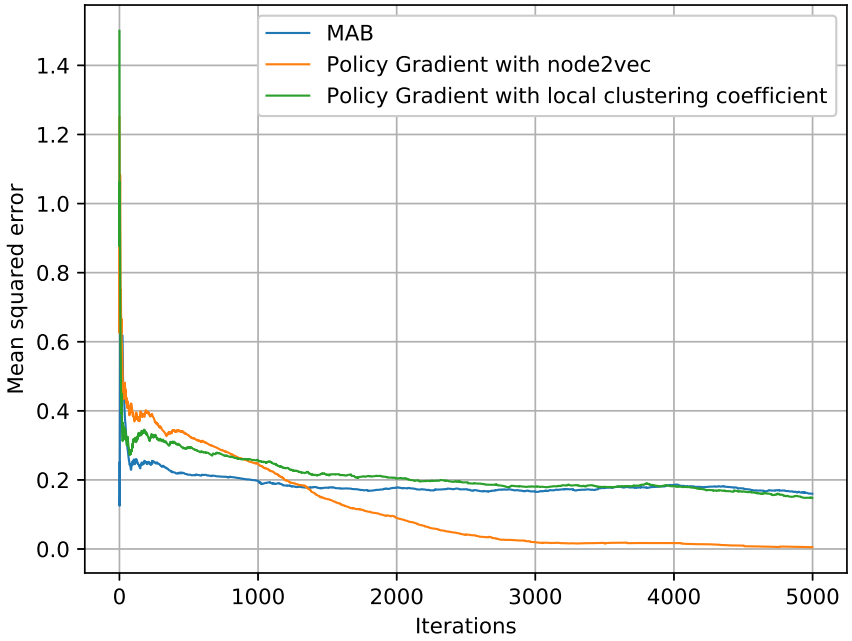


Figure 5.6: Convergence process of the deep RL approaches.

5.4 Experiments with reward choice

In this section we conduct experiments on the applicability of such alternative reward measures as objective function of sparse label propagation and Shannon's entropy of the recovered signal. We generate a graph conforming to the SBM ($p = 0.7$, $q = 0.02$) with 10 clusters each one containing 10 nodes. In each simulation round we sample the graph using URS algorithm and recover it using the convex recovery method (2.2). We then measure mean squared error of graph signal recovery and also note the optimal value of the objective function of the SLP obtained during the current simulation round. This pair of measurements is added to the scatterplot (see Figure 5.8) as one data point. This procedure is repeated for 500 simulation rounds so that the scatterplot accounts for 500 data points in total. We conduct two independent simulations, one with sampling budget $M = 10$ (see Figure 5.8(a)) and the other with sampling budget $M = 20$ (see Figure 5.8(b)).

We also perform similar experiment to one described above, but instead of the optimal value of the objective function of the SLP in each simulation round we measure Shannon's entropy and show it in the scatterplot against the mean squared error of recovery (see Figure 5.9). Again, we do the experiment for two sampling budgets, i.e., $M = 10$ (see Figure 5.9(a)) and $M = 20$

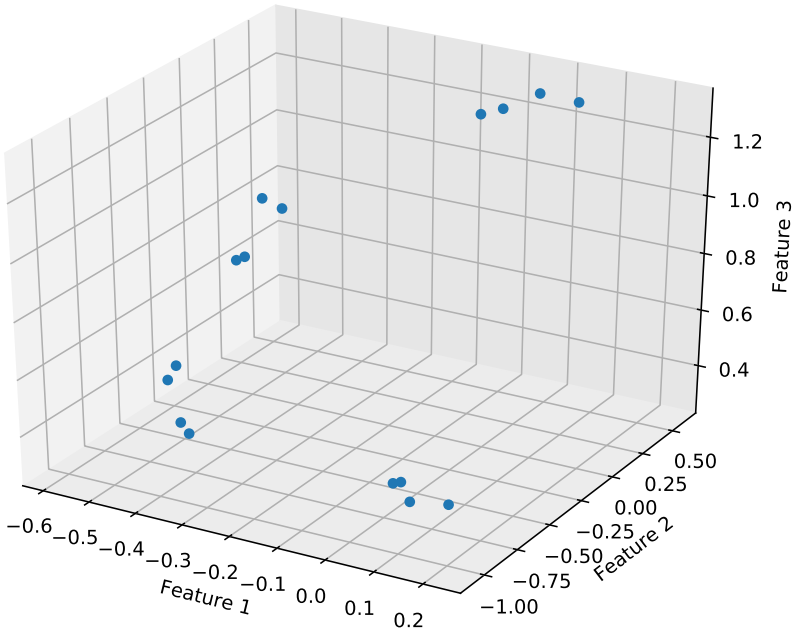
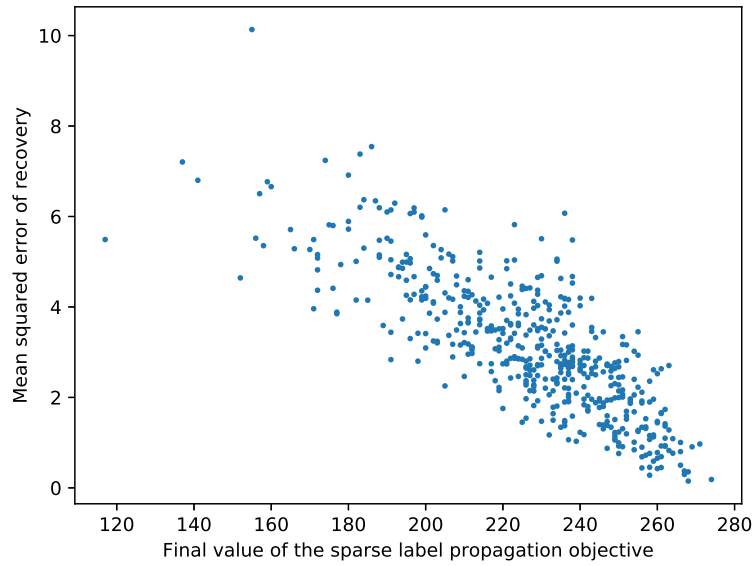
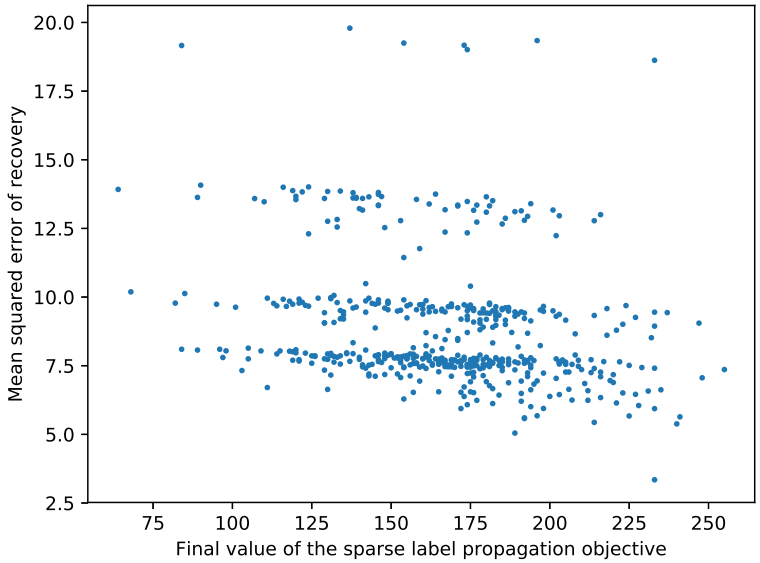


Figure 5.7: Layout of the features in the feature space.

(see Figure 5.9(b)).

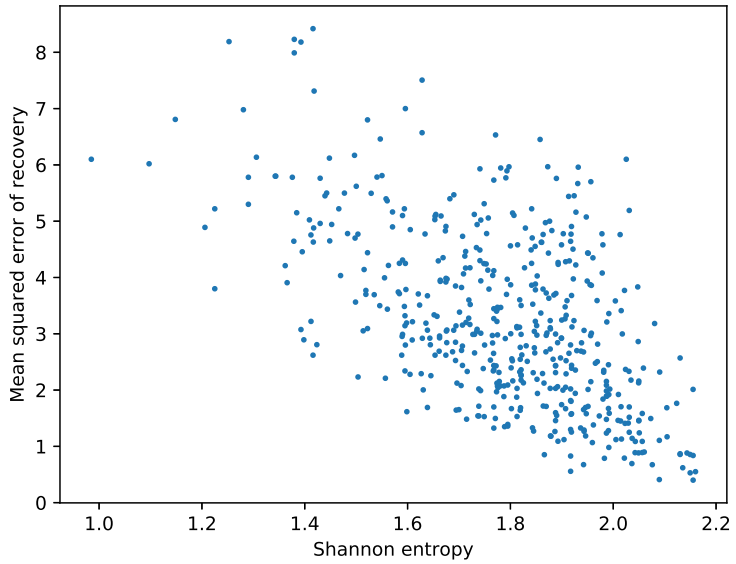


(a)

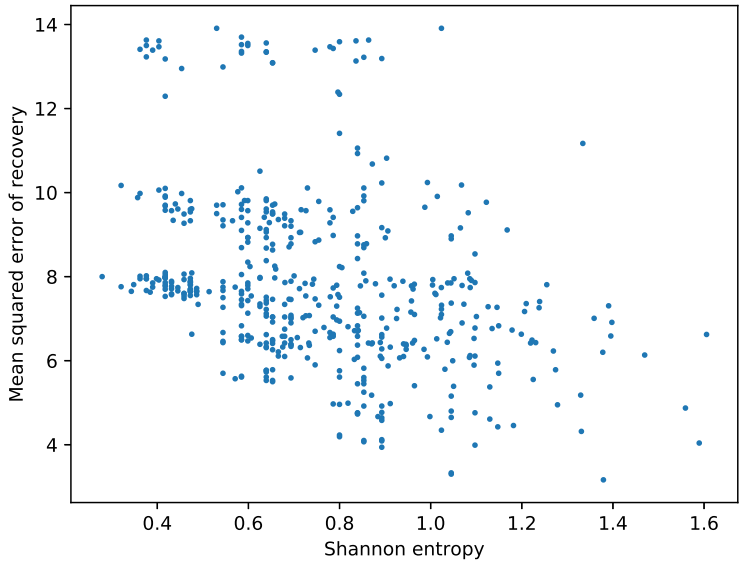


(b)

Figure 5.8: Scatterplot between mean squared error of recovery and optimal value of the SLP optimization objective: (a) – sampling budget $M = 10$, (b) – sampling budget $M = 20$.



(a)



(b)

Figure 5.9: Scatterplot between mean squared error of recovery and Shannon's entropy: (a) – sampling budget $M = 10$, (b) – sampling budget $M = 20$.

Chapter 6

Discussion

We now analyze simulation results presented in the previous chapter and conduct elaborative discussion concerning phenomenons observed. We also define prospects for the future research work and possible developments of the topic.

6.1 Simulation results

6.1.1 General discussion on results

We start with analyzing strengths and weaknesses of the MAB sampling algorithm proposed in the Section 4.1 and simulated in the Section 5.2. First of all, it is interesting to note that in our simulations the algorithm outperforms RWS and URS strategies after 200 and 800 episodes respectively (see Figure 5.2). Convergence speed is high at the initial stage and then substantially decreases after approximately 1000 episodes.

The Figure 5.4 shows that for relative sampling budget 0.2 improvement in NMSE amounts to 5 dBs in comparison to random sampling and 10 dBs in comparison to random walk approach. This gap increases even more for the sampling budget 0.4, to 8 dBs and 20 dBs respectively. The general tendency suggests further increase of the gap for larger sampling budgets.

One of the major disadvantages of the algorithm is that it requires information about the shortest paths between all pairs of nodes, which may not be feasible in the Big Data world due to the computational intractability. Another downside is that the sampling algorithm does not take into account weights of edges. This important information, if handled properly, could potentially improve characteristics of the sampling algorithm. Also, it should be noted that the algorithm is stateless, i.e., does not use any information

about states of the individual nodes, but rather optimizes probability distribution with respect to the global graph features, such as number of clusters, graph diameter and probability of connectivity between nodes.

An attempt to introduce stateful representation for the nodes and move towards deep RL approaches has been made in the Section 4.2 and the appropriate simulations have been conducted in the Section 5.3. There we define policy gradient based approach and simple model of graph containing four cliques. In the Figure 5.6 we compare two variations of policy gradient, with node2vec and clustering coefficient features. For reference, we also present convergence of the MAB-based algorithm. The figure illustrates that the node2vec version has much better performance and is able to find sampling strategy which reconstructs graph perfectly. Performances of the clustering coefficient and MAB versions are similar. This can be explained by the fact that clustering coefficient itself does not provide any meaningful information and, therefore, the algorithm is almost equivalent to the stateless version represented by MAB. It is also interesting to note the layout of the node2vec features in the feature space (see Figure 5.7) where we can easily notice four groups of points corresponding to four clusters of our graph. Intuitively it can be understood that better feature separation in the feature space leads to better performance of the RL algorithm because in this case features provide more information about the underlying model and can be more efficiently used during the learning process.

Another important experiment made within the scope of the current research is related to the reward choice problem. In the Figure 5.8 we depict the scatterplot between mean squared error and the optimization objective of the SLP. It is worth mentioning that for the sampling budget $M = 10$ we observe strictly negative correlation between the aforementioned quantities, i.e., larger optimal value of the optimization objective leads to smaller mean squared error of recovery. At the first glance this may seem strange but as it has been shown by the simple example (c.f. Figure 4.4) small value of the optimization objective may be highly disadvantageous from the standpoint of mean squared error. When the sampling budget increases to $M = 20$ this correlation disappears and it is not possible to reliably establish dependency between these quantities anymore. We then plot similar scatterplots and study dependency between mean squared error and Shannon's entropy (see Figure 5.9). Similarly to the case with objective function, for the sampling budget $M = 10$ we can observe that mean squared error decreases when entropy increases. However, for the case $M = 20$, the dependency deteriorates and does not allow making reliable judgments about the variables correlation.

6.1.2 Note on random walk performance

When analyzing simulation results (see Figure 5.4) of the MAB-based algorithm we can notice that the performance of the RWS is worse than one of its counterparts, i.e., MAB-based sampling and URS. We now discuss this phenomenon and try to explain poor performance of the RWS using a simple argument based on the properties of Markov chains. For simplicity we consider a graph with clusters $\mathcal{C}_1$ and $\mathcal{C}_2$ having sizes N_1 and N_2. The probability of having an edge between nodes in the same cluster is denoted p, while the probability of having an edge between nodes in different clusters is q. An elementary calculation yields the probability of a random walk transitioning from $\mathcal{C}_1$ to $\mathcal{C}_2$ as:

$$p_{12} = \frac{qN_2}{qN_2 + p(N_1 - 1)}$$

Likewise, the probability of staying in the $\mathcal{C}_1$:

$$p_{11} = 1 - p_{12}$$

We note that qN_2 is the expected number of edges between a particular node of $\mathcal{C}_1$ and $\mathcal{C}_2$ and $p(N_1 - 1)$ is the expected number of edges between a particular node of $\mathcal{C}_1$ and the remaining nodes of $\mathcal{C}_1$. Similarly for $\mathcal{C}_2$:

$$p_{21} = \frac{qN_1}{qN_1 + p(N_2 - 1)}$$
$$p_{22} = 1 - p_{21}$$

The transition matrix of a Markov chain, which summarizes probabilistic transitions between clusters, can be formalized as follows:

$$P = \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix}$$

Let $\mathbf{v} = (v_1, v_2)^T$ be an equilibrium distribution [35] of the Markov chain which reflects amount of discrete time spent in $\mathcal{C}_1$ and $\mathcal{C}_2$. According to theory of Markov chains [35] finding this distribution amounts to finding a vector $\mathbf{v}$ such that:

$$\begin{cases} \mathbf{v}^T P = \mathbf{v}^T \\ v_1 + v_2 = 1 \\ v_1 \geqslant 0, v_2 \geqslant 0 \end{cases} \tag{6.1}$$

It is easy to verify that solving the aforementioned system (6.1) yields the following equilibrium distribution:

$$v_1 = \frac{p_{21}}{p_{12} + p_{21}}, v_2 = 1 - v_1$$

We now consider particular example of a random graph with the configuration: $N_1 = 20, N_2 = 80, p = 0.7, q = 0.01$. According to the presented above formulas, computations yield equilibrium distribution: $v_1 \approx 0.05$, $v_2 \approx 0.95$, which means that 95 % of discrete time of a random walk is spent in $\mathcal{C}_2$ whereas only 5% of time is spent in $\mathcal{C}_1$. This rationale implies that upon termination of a random walk instance its endpoints will be located in clusters $\mathcal{C}_1$ and $\mathcal{C}_2$ with probabilities 0.05 and 0.95 respectively.

From the aforementioned examples we can conclude that although $\mathcal{C}_2$ is only four times larger than $\mathcal{C}_1$, the probability of random walk termination within it is larger by a factor ≈ 19. Thus, the random walk sampling algorithm tends to oversample larger clusters and undersample smaller ones. This partially explains the poor performance of random walk in comparison to random sampling which samples clusters proportionally to their sizes.

6.2 Future work

Artificial intelligence in general and reinforcement learning in particular are the areas which can offer variety of fundamentally new approaches for graph signal processing framework. The current study has made an initial attempt to apply these new approaches to the graph signal sampling problem and has revealed obstacles and relevant issues which are awaiting to be resolved. Below we list and briefly discuss, in our opinion, the most important of them.

- **Reward choice.**
  It is unclear what kind of reward measure could be potentially used in real-world settings, because mean squared error, depending on application, may be unavailable. In the current thesis we have tried using Shannon's entropy and optimal value of the sparse label propagation, but none of these has yielded stable correlation with our reference measure, i.e., mean squared error.

- **State and action space choice.**
  In our research we have tried using node2vec feature vectors as states and nodes neighborhoods as actions, even though this selection may be suboptimal. Selection of the appropriate state-action space powerfully determines not only overall degree of success during the training phase but also choice of the reinforcement learning algorithm which can operate on this state-action space in the most efficient way. It is also important to keep in mind possible non-Markovian nature of the transition processes and non-stationarity of the environment which may arise under certain conditions.

- **Online on-policy algorithm for Big Data.**
  Algorithms presented in this research use training subset for learning the policy and test subset for applying it to new, unseen data. For practical use it might be preferable to formulate such a sampling algorithm which would work on huge graphs and in online fashion without using supervised learning approaches.

- **Using only local graph features and information.**
  Both presented in this study algorithms preprocess adjacency matrix to extract information about the shortest paths. However, for large graphs such a preprocessing can be computationally intractable and may not be relied on. Therefore, it is advisable to avoid using any information that requires preprocessing. Simple and efficient RL algorithm should operate on "local view" taking into account information about adjacent nodes only.

- **Choosing appropriate application area.**
  All critical issues mentioned above depend on the actual application area of the graph sampler. For instance, if the problem being solved is an image compression, then it is acceptable to use mean squared error reward as well as some preprocessing. On the other hand, if we are interested in regression problems, this information is rather unavailable because we do not have access to the true graph signal. Generally speaking, overall design of the RL sampling algorithm may vary and powerfully depends on the application area.

Chapter 7

Conclusions

In this master's thesis we have studied applicability of RL to the graph signal sampling problem. The research has encompassed not only theoretical study of the graph signal processing and RL fundamentals but also proposed two novel graph signal sampling methods based on the MAB and policy gradient algorithms. The methods have been tested on the synthetic datasets. We have also discussed obstacles reducing the efficiency of the presented RL solutions and drawn prospects for the future work.

On the practical side, we have strengthened knowledge of the foundations of graph signal processing, machine learning, RL and have practiced implementing modern deep RL algorithms using popular Tensorflow library.

Bibliography

[1] Abramenko, O., and Jung, A. Graph signal sampling via reinforcement learning. *arXiv preprint arXiv:1805.05827* (2018).

[2] Ambos, H., Tran, N., and Jung, A. The Logistic Network Lasso. *ArXiv e-prints* (May 2018).

[3] Basirian, S., and Jung, A. Random walk sampling for big data over networks. In *Sampling Theory and Applications (SampTA), 2017 International Conference on* (2017), IEEE, pp. 427–431.

[4] Bellman, R. *Dynamic programming*. Courier Corporation, 2013.

[5] Besbes, O., Gur, Y., and Zeevi, A. Stochastic multi-armed-bandit problem with non-stationary rewards. In *Advances in neural information processing systems* (2014), pp. 199–207.

[6] Bian, F., Kempe, D., and Govindan, R. Utility based sensor selection. In *Proceedings of the 5th international conference on Information processing in sensor networks* (2006), ACM, pp. 11–18.

[7] Chambolle, A., and Pock, T. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of mathematical imaging and vision 40*, 1 (2011), 120–145.

[8] Chamon, L. F., and Ribeiro, A. Greedy sampling of graph signals. *IEEE Transactions on Signal Processing 66*, 1 (2018), 34–47.

[9] Chen, S., Varma, R., Singh, A., and Kovačević, J. Signal recovery on graphs: Fundamental limits of sampling strategies. *IEEE Transactions on Signal and Information Processing over Networks 2*, 4 (2016), 539–554.

[10] Decelle, A., Krzakala, F., Moore, C., and Zdeborová, L. Asymptotic analysis of the stochastic block model for modular networks and its algorithmic applications. *Physical Review E 84*, 6 (2011), 066106.

[11] Goldberg, Y., and Levy, O. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722* (2014).

[12] Grant, M., and Boyd, S. CVX: Matlab software for disciplined convex programming, version 2.1. `http://cvxr.com/cvx`, Mar. 2014.

[13] Grover, A., and Leskovec, J. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining* (2016), ACM, pp. 855–864.

[14] Hallac, D., Leskovec, J., and Boyd, S. Network lasso: Clustering and optimization in large graphs. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining* (2015), ACM, pp. 387–396.

[15] Heimlicher, S., Lelarge, M., and Massoulié, L. Community detection in the labelled stochastic block model. *arXiv preprint arXiv:1209.2910* (2012).

[16] Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. Rainbow: Combining improvements in deep reinforcement learning. *arXiv preprint arXiv:1710.02298* (2017).

[17] Howard, R. A. Dynamic programming and markov processes.

[18] Joshi, S., and Boyd, S. Sensor selection via convex optimization. *IEEE Transactions on Signal Processing 57*, 2 (2009), 451–462.

[19] Jung, A. Learning the conditional independence structure of stationary time series: A multitask learning approach. *IEEE Transactions on Signal Processing 63*, 21 (2015), 5677–5690.

[20] Jung, A., Berger, P., Hannak, G., and Matz, G. Scalable graph signal recovery for big data over networks. In *Signal Processing Advances in Wireless Communications (SPAWC), 2016 IEEE 17th International Workshop on* (2016), IEEE, pp. 1–6.

[21] Jung, A., Hannak, G., and Goertz, N. Graphical lasso based model selection for time series. *IEEE Signal Processing Letters 22*, 10 (2015), 1781–1785.

[22] Jung, A., Hero III, A. O., Mara, A., and Jahromi, S. Semi-supervised learning via sparse label propagation. *arXiv preprint arXiv:1612.01414* (2016).

[23] Jung, A., and Hulsebos, M. The network nullspace property for compressed sensing of big data over networks. *Front. Appl. Math. Stat. — doi: 10.3389/fams.2018.00009* (2018).

[24] Jung, A., Tran, N., and Mara, A. When is network lasso accurate? *Front. Appl. Math. Stat. — doi: 10.3389/fams.2018.00009 3* (2018), 28.

[25] Jung, A., Tran Quang, N., and Mara, A. When is network lasso accurate? *Frontiers in Applied Mathematics and Statistics 3* (2017), 28.

[26] Kakade, S. M. A natural policy gradient. In *Advances in neural information processing systems* (2002), pp. 1531–1538.

[27] Lawler, E. L., and Wood, D. E. Branch-and-bound methods: A survey. *Operations research 14*, 4 (1966), 699–719.

[28] Levine, S., Finn, C., Darrell, T., and Abbeel, P. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research 17*, 1 (2016), 1334–1373.

[29] Li, J., Monroe, W., Ritter, A., Galley, M., Gao, J., and Jurafsky, D. Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541* (2016).

[30] Liben-Nowell, D., and Kleinberg, J. The link-prediction problem for social networks. *Journal of the American society for information science and technology 58*, 7 (2007), 1019–1031.

[31] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[32] Mara, A., et al. A comparative analysis of graph signal recovery methods for big data networks.

[33] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[34] Mossel, E., Neeman, J., and Sly, A. Stochastic block models and reconstruction. *arXiv preprint arXiv:1202.1499* (2012).

[35] Norris, J. R. *Markov chains*. No. 2. Cambridge university press, 1998.

[36] Quang, N. T., and Jung, A. On the sample complexity of graphical model selection for non-stationary processes. *arXiv preprint arXiv:1701.04724* (2017).

[37] Romero, D., Ma, M., and Giannakis, G. B. Kernel-based reconstruction of graph signals. *IEEE Transactions on Signal Processing 65*, 3 (2017), 764–778.

[38] Schaback, R. A practical guide to radial basis functions. *Electronic Resource 11* (2007).

[39] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. Deterministic policy gradient algorithms. In *ICML* (2014).

[40] Sutton, R. S., and Barto, A. G. *Reinforcement learning: An introduction, second edition (complete draft)*, vol. 1. MIT press Cambridge, 2017.

[41] Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems* (2000), pp. 1057–1063.

[42] Tan, M., Shi, Q., van den Hengel, A., Shen, C., Gao, J., Hu, F., and Zhang, Z. Learning graph structure for multi-label image classification via clique generation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2015), pp. 4100–4109.

[43] Tanaka, Y. Spectral domain sampling of graph signals. *IEEE Transactions on Signal Processing* (2018).

[44] Tieleman, T., and Hinton, G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning 4*, 2 (2012), 26–31.

[45] Van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *AAAI* (2016), vol. 16, pp. 2094–2100.

[46] Venkitaraman, A., Chatterjee, S., and Haändel, P. Gaussian processes over graphs. *arXiv preprint arXiv:1803.05776* (2018).

[47] Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., and De Freitas, N. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581* (2015).

[48] Watkins, C. J., and Dayan, P. Q-learning. *Machine learning 8*, 3-4 (1992), 279–292.

[49] Xie, X., Feng, H., Jia, J., and Hu, B. Design of sampling set for bandlimited graph signal estimation. In *2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)* (Nov 2017), pp. 653–657.

[50] Yang, Y., Lad, A., Shu, H., Kisiel, B., Cumby, C. M., Ghani, R., and Probst, K. Graph structure learning for task ordering. In *ICEIS (2)* (2009), pp. 164–169.

[51] Yao, L., Sethares, W. A., and Kammer, D. C. Sensor placement for on-orbit modal identification via a genetic algorithm. *AIAA journal 31*, 10 (1993), 1922–1928.

[52] Zhang, D., Maei, H., Wang, X., and Wang, Y.-F. Deep reinforcement learning for visual object tracking in videos. *arXiv preprint arXiv:1701.08936* (2017).

[53] Zhu, X., and Ghahramani, Z. Learning from labeled and unlabeled data with label propagation.

[54] Zhu, Y., Mottaghi, R., Kolve, E., Lim, J. J., Gupta, A., Fei-Fei, L., and Farhadi, A. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on* (2017), IEEE, pp. 3357–3364.